repository.ub.ac.id

IMPLEMENTASI *PURE DATA PROGRAMMING* UNTUK *GENERATE REAL-TIME GAME AUDIO SYNTHESIS* PADA *PLATFORMER GAME*

SKRIPSI

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Bariq Najmi Rizqullah Kartiko Putro
NIM: 145150207111089

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

UNIVERSITAS BRAWIJAYA

PENGESAHAN

IMPLEMENTASI *PURE DATA PROGRAMMING* UNTUK *GENERATE REAL-TIME GAME AUDIO SYNTHESIS* PADA *PLATFORMER GAME*

SKRIPSI

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Bariq Najmi Rizqullah Kartiko Putro
NIM: 145150207111089

Skripsi ini telah diuji dan dinyatakan lulus pada
2 Agustus 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Eriq Muhammad Adams Jonemaro, S.T, M.Kom	Tri Afirianto, S.T, M.T
NIP: 19850410 201212 1 001	NIK: 201309 851213 1 001

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D
NIP: 19710518 200312 1 001

repository.ub.ac.id

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 2 Agustus 2018

Bariq Najmi Rizqullah Kartiko Putro

NIM: 145150207111089

iii

UNIVERSITAS BRAWIJAYA

KATA PENGANTAR

Puji dan syukur penulis panjatkan kehadirat Allah Subhanahu Wa Ta'ala atas segala Rahmat dan Karunia-Nya, sehingga skripsi yang berjudul "Implementasi *Pure Data Programming* Untuk *Generate Real-Time Audio Synthesis* Pada *Platformer Game*" dapat diselesaikan dengan baik dan tepat waktu.

Dalam kesempatan ini saya selaku penulis skripsi menyampaikan rasa terima kasih kepada semua pihak yang telah membantu menyumbangkan ide, pikiran, serta dukungan selama penyusunan dan penelitian skripsi ini, diantaranya :

1. Keluarga yaitu kedua orang tua saya (Ayah dan Bunda) beserta adik dan kakak saya yang selalu memberi dukungan serta doa.
2. Bapak Eriq Muhammad Adams Jonemaro, S.T, M.Kom dan Bapak Tri Afirianto, S.T., M.T. selaku Dosen Pembimbing I dan II yang telah meluangkan waktu untuk membimbing dan mengarahkan penulis dalam penulisan skripsi ini hingga selesai.
3. Teman-teman kontrakan TPOC atas bantuan, motivasi dan kebersamaan selama diperkuliahan dan diluar perkuliahan hingga dapat menyelesaikan skrpsi ini dengan baik.

Penulis menyadari bahwa dalam penyusunan skripsi ini masih banyak kekurangan, sehingga saran dan kritik yang membangun dapat disampaikan secara langsung untuk pengembangan dan penelitian selanjutnya.

Malang, 2 Agustus 2018

Penulis

bariq3396@gmail.com

repository.ub.ac.id

ABSTRAK

Audio memainkan peran yang sangat penting di dalam suatu permainan selain dari komponen lain pengembangan sebuah permainan. Secara tradisional, teknologi *audio* dihasilkan dengan proses perekaman atau *recording*. Dengan menggunakan proses perekaman, hasil yang didapatkan adalah serangkaian gelombang amplitudo, yang mana mengandung suara-suara dari hasil perekaman. Hasil dari perekaman tersebut menghasilkan suatu data. Ruang penyimpanan yang dibutuhkan untuk menyimpan sebuah data suara yaitu, setiap 1 menit dari suara *stereo* dengan *sample rate* 44,1 kHz membutuhkan 10 *megabyte* ruang penyimpanan. Terdapat teknologi *audio* yang dapat dimanfaatkan dalam pengembangan permainan. *Pure Data* merupakan bahasa pemrograman visual untuk tujuan multimedia. *Pure Data* dapat digunakan untuk proses dan menghasilkan suara, *video*, grafis dua Dimensi atau tiga dimensi, dan sebuah MIDI (Musical Intrument Digital Interface) dalam kepentingan insinyur *audio*. Selain itu, terdapat kerangka kerja alat bantu *Heavy* untuk menghasilkan *audio* tambahan menjembatani implementasi dari *Pure Data* ke *Unity*. *Heavy* menggunakan prinsip perangkat lunak modern untuk menghasilkan kode. Pada penelitian kali ini dibahas mengenai efek suara dan musik latar. Semua aset suara yang dihasilkan akan diimplementasikan ke dalam permainan. Hasil dari penelitian ini adalah perbandingan ukuran dokumen dari ketiga jenis suara yaitu, efek suara, musik latar, dan musik menu utama. *Pure Data* menghasilkan prosedural *audio* yang merupakan jenis *audio* dinamis. Kelebihan dari *audio* dinamis adalah efek suara yang dihasilkan memberikan kontribusi keterlibatan emosional kepada pemain. Kerangka kerja *Heavy* digunakan untuk mengimplementasikan *Pure Data* ke *Unity*. Terdapat dua pengujian untuk menguji *Pure Data* yaitu, dengan pengujian unit dan validasi.

Kata kunci: *Pure Data*, *Unity*, *Heavy Audio Tools*, Efek Suara, Musik Latar

UNIVERSITAS BRAWIJAYA

repository.ub.ac.id

ABSTRACT

Audio plays a very important role in the game besides the another component of game development. Traditional audio technology has its foundations in recording. By using the recording processs, the result obtained are a series of amplitude waves, which contain sounds from recording. The result of the recording produce a data. The storage space needed to store a sound data is every 1 minute of stereo sound with a 44,1 kHz sample rate requires 10 megabyte of storage space. There is an audio technology that can be used in game development. Pure Data is visual programming language for multimedia purpose. Pure Data can be used to process and generate sound, video, 2D or 3D graphics and MIDI for sound engineering purpose. Furthermore, there is framework called Heavy for easily generating audio plugins, used in implementating Pure data to Unity. Heavy make use of modern software pronciples to generate highly optimized code. In this study discussed the sound effects and background music. Sound assets will be implemented into the game. The result of this study are the comparison of document sizes of the three types of sound, sound eefect, background music, and main menu music. Pure Data generate procedural audio which is a dynamic type of audio. The advantage of dynamic audio is sound effects that contribute to emotional involvement of the player. Heavy framework is used to implement Pure Data to unity. There are two types of testing Pure Data, unit testing and validation.

Keywords: *Pure Data*, *Unity*, *Heavy Audio Tools*, *Sound Effect*, *Background Music*

UNIVERSITAS BRAWIJAYA

DAFTAR ISI

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

DAFTAR TABEL

DAFTAR GAMBAR

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

BAB 1 PENDAHULUAN

1.1 Latar belakang

Audio yang terdapat di dalam suatu *video game* merupakan fitur interaktif yang dipresentasikan, yaitu sebagai aksi yang memicu keluarnya sebuah dialog, efek suara, *ambient music*, hingga musik latar. Secara tradisional teknologi audio dihasilkan dengan proses perekaman atau *recording*. Sinyal suara di dunia nyata ditangkap melalui mikrofon, selanjutnya terdapat proses *mixing*, dan *mastering* untuk dirubah kedalam bentuk akhir dari *audio* tersebut (Farnell, 2007). Setiap bagian musik dan efek suara dipastikan diciptakan melalui langkah ini. Dengan proses perekaman, hasil yang didapatkan adalah serangkaian gelombang amplitudo. Gelombang tersebut mengandung suara-suara dari hasil perekaman dan hasil perekaman tersebut menghasilkan suatu data. Contohnya adalah data *audio* dengan format *mp3*. Banyak ruang penyimpanan yang dibutuhkan untuk menyimpan sebuah data suara yaitu, setiap 1 menit dari suara *stereo* dengan *sample rate* 44,1 kHz membutuhkan 10 *megabyte* ruang penyimpanan (Gamasutra, 1997).

Terdapat alternatif yang dapat memudahkan pembuatan *audio*, suara orisinal dan kreasi musik, di dalam *video game* dalam bentuk teknologi lingkungan pengembangan terintegrasi. *Pure Data* merupakan salah satu teknologi *audio* yang handal serta cakupan penggunanya cukup luas. *Pure Data* adalah bahasa pemrograman visual yang *open source* ditujukan untuk multimedia menjadi sebuah lingkungan pengembangan sehingga dapat digunakan sebagai *sound engine* di dalam aplikasi *mobile*, permainan, halaman sebuah *web*, dan proyek seni. Penggunaan dari *Pure Data* sangatlah mudah dan sederhana, selain itu dapat berjalan fleksibel sesuai kapabilitas *Pure data* itu sendiri. Contoh seberapa fleksibel *Pure Data* adalah penerapan fungsinya di dalam *mobile apps* seperti *iOS* dan *Android*, hingga musik interaktif di dalam pengembangan *game* (Kirn, 2012).

Pure Data memberi kemudahan dalam menghasilkan *audio*. Pure *Data* mempunyai kelebihan dibandingkan dengan teknologi *audio* tradisional lainnya. *Pure Data* menghasilkan prosedural *audio* yang merupakan jenis audio yang dinamis (*dynamic*). *Audio* dinamis dapat didefinisikan menjadi dua bentuk yang berbeda yaitu interaktif (*interactive*) *audio* dan adaptif (*adaptive*) *audio*.

Penelitian ini membahas tentang implementasi bahasa pemrograman visual *Pure Data* yang dimana pada penelitian ini difokuskan untuk pembuatan prosedural *audio* di dalam studi kasus *video game* yang ber*genre Platformer* yang mana mekanik dalam permainan ini pemain membimbing atau mengontrol karakter untuk melompati panggung (*platform*), hambatan, dan rintangan (Greenslade, 2006). Jenis suara yang difokuskan pada penelitian ini adalah *synthesis*. *Synthesis* sendiri merupakan proses dari penggabungan kumpulan acak suara elektronik yang nantinya memiliki keluaran suara yang beragam sesuai dengan kreativitas pembuatnya (Russ, 2009).

Hasil dari penelitian ini berupa *demo* permainan yang memanfaatkan *Pure data* dan kerangka kerja (*framework*) *Heavy* dalam pembuatan prosedural *audio* yang terkandung di dalamnya. Diharapkan dengan adanya penelitian ini penulis ikut menambah kajian yang lebih difokuskan ke arah *audio* permainan di dalam pengembangan sebuah *video game*.

1.2 Rumusan masalah

Berdasarkan uraian latar belakang diatas, maka didapatkan beberapa rumusan masalah terkait penelitian sebagai berikut :

1. Bagaimana cara untuk menghasilkan *real-time game audio*?
2. Bagaimana perbandingan ukuran dokumen dari hasil implementasi *Pure Data* dengan *audio* yang dihasilkan dari proses perekaman?
3. Bagaimana mengimplementasikan *Pure Data* ke *Unity* sehingga dapat dimanfaatkan dalam pengembangan permainan *platformer*?
4. Bagaimana hasil dari pengujian fungsionalitas pada *Pure Data*?

1.3 Tujuan

Di dalam penelitian ini bertujuan untuk menunjukkan bahwa terdapat cara lain dalam merancang, menerapkan, dan membuat *audio*, musik, serta efek suara (*sound effect*) oleh desainer suara untuk mendapatkan hasil yang berbeda.

1. Mengetahui terdapat teknologi *Pure data* yang dapat dimplementasikan dalam membuat prosedural *audio* atau pembuatan *game audio* secara *real-time* untuk kepentingan pengembangan dalam suatu proyek *video game*.
2. Mengetahui perbandingan ukuran dokumen hasil dari implementasi *Pure Data* dengan *audio* yang dihasilkan dari perekaman.
3. Menerapkan kerangka kerja (*framework*) *Heavy* yang merupakan perangkat pembantu *audio* untuk menjembatani dalam menghasilkan dan memanipulasi *audio* permainan ber*genre platformer* di dalam Unity.
4. Mensimulasikan hasil prosedural *audio* ke dalam *video game* ber*genre platformer* untuk melihat *valid* atau tidak *valid* hasil dari fungsionalitas *Pure Data*.

1.4 Manfaat

Berikut ini merupakan manfaat dari penelitian yang dilakukan :

1. Bagi penulis, menambah wawasan baru dalam dunia *sound designer.*
2. Bagi pengembang *game*, menambah referensi dalam pembuatan *audio* yang memiliki keunggulan dalam penerapannya yaitu *audio* yang berjalan secara *real-time* atau dapat dikatakan secara prosedural *audio*.

3. Bagi pemain, mendapatkan pengalaman bermain dan merasakan perbedaan dari dalam atmosfir permainan yang tercipta karena suara yang dihasilkan memanfaatkan teknologi *audio*.

1.5 Batasan masalah

Batasan pada penelitian ini adalah :

1. *Genre* permainan pada penelitian ini adalah *platformer*.
2. *Platform* pada penelitian ini adalah berbasis *Pc Windows*.
3. Jenis suara pada penelitian ini adalah *synthesis*.
4. Penelitian ini hanya membahas mengenai teknologi *Pure Data* untuk pembuatan prosedrual *audio* pada sebuah *video game*.
5. Penelitian ini hanya difokuskan pada *Pure Data* yang akan diterapkan pada mesin permainan Unity.

1.6 Sistematika pembahasan

Sistematika pembahasan secara garis besar meliputi beberapa bab :

BAB I : PENDAHULUAN

Bab ini membahas mengenai latar belakang, identifikasi masalah, rumusan masalah, tujuan, manfaat, sistematika pembahasan dan jadwal pada penelitian.

BAB II : LANDASAN KEPUSTAKAAN

Bab ini berisi tentang teori-teori dan referensi yang mendukung dalam perancangan, implementasi dan pengujian dalam metode yang digunakan.

BAB III : METODOLOGI

Bab ini membahas mengenai metode yang akan diimplementasikan dalam penelitian yang meliputi dasar teori, metode perancangan, metode implementasi dan metode pengajuan serta pengambilan kesimpulan, saran.

BAB IV : PERANCANGAN

Bab ini memuat perancangan *Pure Data* yang terdiri dari perancangan efek suara dan musik latar yang nantinya akan diterapkan ke dalam Unity.

BAB V : IMPLEMENTASI

Bab ini memuat implementasi dari simulasi *platformer game* menggunakan teknologi *Pure data* di dalam *game engine* Unity.

BAB VI : PENGUJIAN

Bab ini memuat hasil pengujian dari hasil implementasi yang sudah dibuat.

BAB VII : PENUTUP

Bab terakhir memuat kesimpulan yang didapatkan dari pembahasan serta saran untuk penelitian kedepannya.

BAB 2 LANDASAN KEPUSTAKAAN

2.1 *Pure Data*

Pure Data (atau hanya *Pd*) adalah sebuah lingkungan bahasa pemrograman *visual* untuk urusan multimedia yang bersifat *open source* atau bebas dalam penggunaanya dan dapat berjalan dimana saja mulai dari komputer personal hingga *embedded device* atau gawai tanam bahkan ponsel pintar. *Pure Data* merupakan salah satu penghasil prosedural *audio*, yang memiliki artian yaitu proses pembuatannya dilakukan secara langsung (*real-time*) berdasarkan kumpulan aturan pemrograman dan sebuah masukan (*input*) (Farnell, 2007). Proses harus diaplikasikan dengan interval waktu yang menjamin bahwa sinyal latensi dari masukan ke keluaran tidak dapat diterima atau dirasakan oleh pendengaran manusia (Ausin et al., 2017).

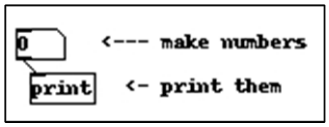
Prosedural *audio* sendiri adalah bentuk non-linear atau sistem yang tidak memiliki ketetapan, seringkali penerapannya ke arah jenis suara synthetis, dan dibuat secara langsung atau *real-time* sesuai dengan aturan pemrograman yang ada. Dapat dikatakan bahwa prosedural *audio* adalah *computer sound*. Prosedural *audio* secara istilah merupakan sistem yang berfungsi dalam waktu nyata (*real-time*). Sifat dari prosedural *audio* tidak dihasilkan dari proses perekaman, dimana program dijalankan, biasanya dengan beberapa parameter untuk menentukan jenis keluaran yang diinginkan, dan secara langsung memulai menghasilkan suara dan musik yang berubah secara terus-menerus sepanjang waktu, atau merespon masukan dengan membuat suara dan musik (Farnell, 2007).

Pure Data merupakan percabangan dari bahasa pemrograman *patcher* yang diketahui sebagai *Max*. Pendistribusian utamanya dikembangkan oleh Miller Puckette. *Pure Data* didistribusikan dengan cara diubah ke dalam bentuk GUI serta diperbanyak ekternal *library* yang termasuk di dalamnya. *Pure Data* dapat membuat musisi, artis visual, seorang penampil, peneliti, hingga pengembang untuk menciptakan perangkat lunak secara gambar (*visual*), tidak dengan menulis sekumpulan baris kode. *Pure Data* dapat digunakan untuk memproses dan menghasilkan suara, *video*, grafis dua dimensi/tiga dimensi, dan sensor antarmuka, masukan atau *input* dari sebuah alat atau *device*, dan MIDI (*Musical Intrument Digital Interface*). Di dalam *Pure Data* tersendiri, kita dapat menciptakan suara *synthesis* yang terkustomisasi, efek suara, pola musikal, hingga mesin musikal dengan mengkoneksikan obyek-obyek yang terdapat di dalam layar. *Pure Data* memiliki kemudahan dalam mempelajari kustomisasi untuk sebuah musik dan pengembangan dalam ranah suara.

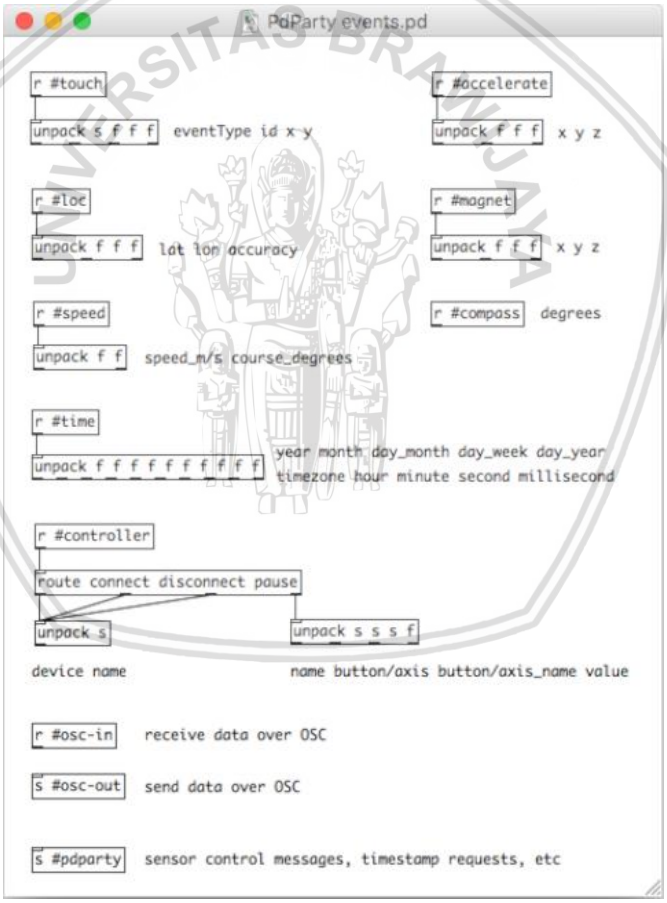
2.1.1 *Pure Data Patch*

Dokumen yang telah dibuat dari *Pure Data* dinamakan *patch*. Setiap dokumen memiliki satu jendela utama dan beberapa sub jendela. Sub jendela tersendiri dapat dibuka dan ditutup tetapi akan selalu berjalan apakah terlihat ataupun tidak terlihat. Pada Gambar 2.1 merupakan contoh sederhana sebuah *Pure Data patch*.

Terdapat dua *box* atau kotak di dalam *patch* tersebut. Kotak *number* dan kotak *object* terkoneksi satu sama lain, keluaran dari kotak *number* terhubung ke masukan kotak *object*. Kotak ini memungkinkan untuk tidak memiliki atau bahkan memiliki banyak masukan maupun keluaran. Letak masukan atau *input* terdapat diatas kotak sedangkan keluaran atau *output* terletak dibawah kotak. Pada Gambar 2.2 merupakan contoh lain penerapan nyata *Pure Data patch* karangan Dan Wilcox yang mana *patch* disini adalah kumpulan obyek yang sudah diprogram sesuai keinginan dan dihubungkan ke setiap obyek. Sehingga dapat dikatakan *patch* adalah sebuah program yang telah diselesaikan dari *Pure Data*.



Gambar 2.1 *Pure Data patch* sederhana
Sumber : Pure Data (2018)



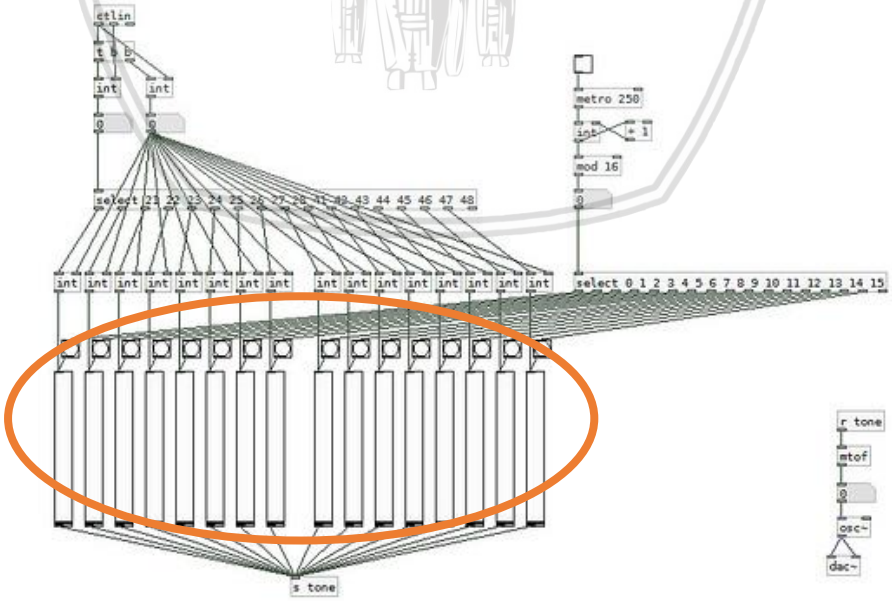
Gambar 2.2 *Pure Data Patch* dalam studi kasus PdParty karangan Dan Wilcox.
Sumber : Dan Wilcox (2006)

2.1.2 *Sequencer*

Sequencer secara umum adalah alat perangkat keras atau perangkat lunak yang dapat merekam, menerima atau mengirim, mengolah bunyi digital dari suatu

intrumen musik elektronik (Widodo, 2006). Dapat dikatakan di dalam sebuah *Pure Data patch*, *sequencer* merupakan istilah bagian program yang terdiri dari kumpulan obyek *Pure Data* sebagai alat pengolah, pengirim atau penerima, hingga perekam bunyi digital. Pada gambar 2.3 merupakan contoh perangkat keras *Pads*. *Pads* jenis ini dapat digunakan untuk membuat *electronic music*. *Pads* ini memiliki enam belas tombol putar yang berfungsi sebagai pusat kontrol *sequencer*. Pada gambar 2.4 diaplikasikan *Pure Data patch* yang mewakili keenam belas *sequencer* tersebut.

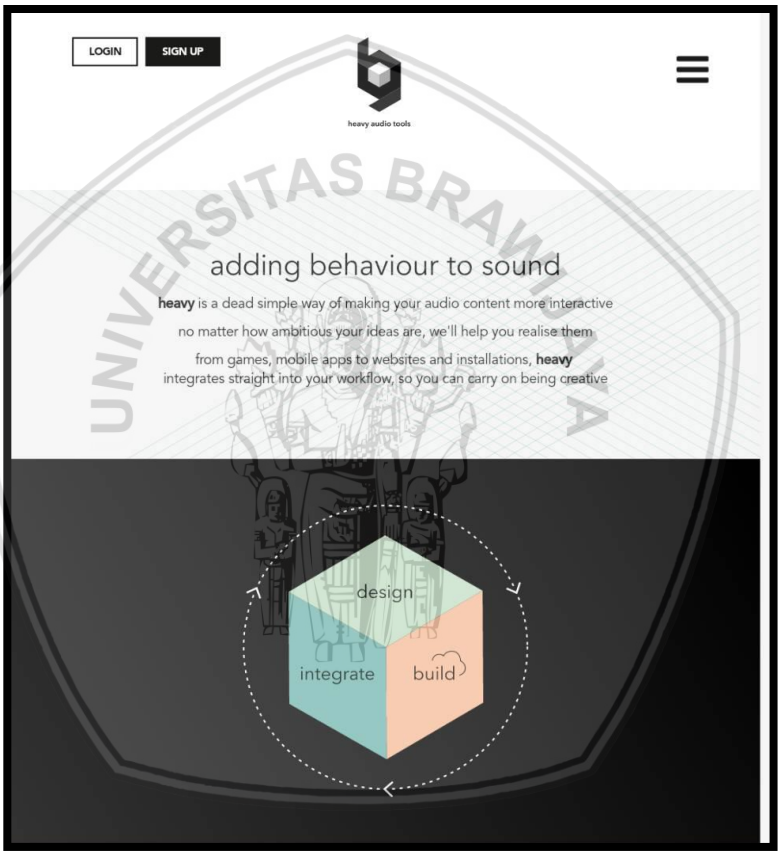
Gambar 2.3 *Novation Lauch Control* dengan *16-step sequencer*
Sumber : Novation (2018)



Gambar 2.4 Sebuah *Pure Data patch* dengan *16-step sequencer*
Sumber : Gamasutra (2018)

2.1.3 *Pure Data* dengan *Heavy*

Heavy merupakan kerangka kerja atau *framework* untuk memudahkan dalam menghasilkan *plugin audio* atau fungsionalitas tambahan sebuah *audio*. Contoh penggunaan dari *plugin audio* tersebut adalah suara interaktif dan aplikasi musik seperti permainan, hingga instrumen musik tertentu. *Heavy* memanfaatkan prinsip perangkat lunak modern untuk menghasilkan kode C/C++ yang sangat teroptimasi, secara spesifik menyasar jenis-jenis populer arsitektur perangkat keras dan kerangka kerja sebuah perangkat lunak seperti *Unity*. *Heavy* dapat menginterpretasikan dan mengubah sekumpulan fitur dari *Pure Data patch*. Heavy tidak dapat memanfaatkan seluruh penggunaan kode *Pure Data* dan sepenuhnya tidak terikat oleh *Pure Data engine*.



Gambar 2.5 Tampilan sekilas Laman Kerangka Kerja *Heavy*
Sumber : Enzien Audio (2018)

2.2 *Platformer Game*

Platformers atau permainan *platform* adalah jenis permainan tentang karakter yang digerakan oleh pemain (*player*), yang mana tujuannya adalah lari dan melompat untuk menghindari *obstacle* dan/atau untuk mengalahkan musuh. *Platformer* seringkali diklasifikasikan sebagai *sub-genre* dari permainan aksi, dan dianggap menjadi *genre game* pertama yang pernah ada. Meskipun menjadi *genre game* pertama, permainan *platform* dipertahankan kepopulerannya dari tahun ke tahun (Minkkinen, 2016).

Level adalah ruang yang mana pemain (*player*) mengeksplorasi *rules* atau aturan serta mekanik yang terdapat pada permainan. Desain *level* yang baik merupakan hal vital untuk membuat permainan yang dapat dinikmati (Byrne, 2005). Level menyediakan ruang interaktif dan kemampuan terhadap pemain untuk mengekplorasi aturan di dalam dunia *game*. *Level design* merupakan pekerjaan yang rumit, untuk menghasilkannya membutuhkan pemahaman semua komponen sebuah permainan dan bagaimana menjadikan komponen tersebut menjadi kesatuan. Di bawah ini merupakan kategori komponen dari *platformer level* sesuai perannya di dalam sebuah *level* permainan sebagai berikut.

1. *Avatar*

   Avatar merujuk pada karakter yang dikendalikan oleh pemain (*player*), meskipun biasanya terdapat banyak pilihan karakter yang dapat dimainkan, pemain biasanya hanya dapat mengendalikan satu karakter dalam suatu waktu. *Avatar* dipilih tidak hanya karena penampilannya, akan tetapi sering kali hal penting yang membuatnya dipilih adalah perbedaan tingkah laku dari karakter tersebut. Pada Gambar 2.6 *avatar* berbentuk naga hijau kecil merupakan Yoshi pada permainan berjudul *Yoshi's Island DS*, pemain dapat menggerakan *avatar* tersebut dengan bebas. Terkadang *avatar* dapat digantikan dalam beberapa waktu selama *level* permainan berlangsung. Contohnya pada *"stork stations"* di dalam *Yoshi's Island DS*. Kemampuan untuk merubah karakter pada tengah permainan memperkenalkan sebuah aspek *puzzle* ke dalam permainan, seperti pada Gambar 2.6 yang mana pemain harus memilih karakter tertentu sehingga Yoshi dapat mengambang dengan angin.

   Avatar secara spesifik memiliki bermacam-macam kemampuan untuk berjalan yang disesuaikan oleh jenis permainannya. Akan tetapi semua *avatar* mempunyai kontrol atas pergerakan horizontal dan setidaknya control terbatas dari pergerakan vertikal, biasanya bentuk dari melompat atau berjongkok. Lebih lanjutnya kemampuan seperti dua kali melompat atau dapat melompati tembok. Jenis *platformer* aksi sering kali menyediakan *avatar* dengan peralatan untuk menghancurkan musuh.

2. *Platform*

   Platform didefinisikan sebagai obyek yang mana *avatar* dapat melewatinya dengan berjalan atau pun melarikan diri secara aman. Dalam suatu *scene*, obyek tersebut disiapkan untuk tujuan lainnya, contohnya seperti suatu benda berbentuk *box* di dalam game *Super Mario World*. *Platform* selalu mempunyai atribut berupa friksi atau geseran, ukuran, dan lereng. Gambar 2.7 adalah contoh *platform* berwarna coklat yang pergerakannya mengikuti jalan yang bertanda garis hitam tipis. *Platform* sering kali dibentuk, direncanakan, dan disusun oleh *level designer*.

Gambar 2.6 Avatar pada permainan Yoshi's Island DS
Sumber : Gillian Smith et al. (2008)

Gambar 2.7 Landasan (*Platform*) dari permainan *Super Mario World*
Sumber : Gillian Smith et al. (2008)

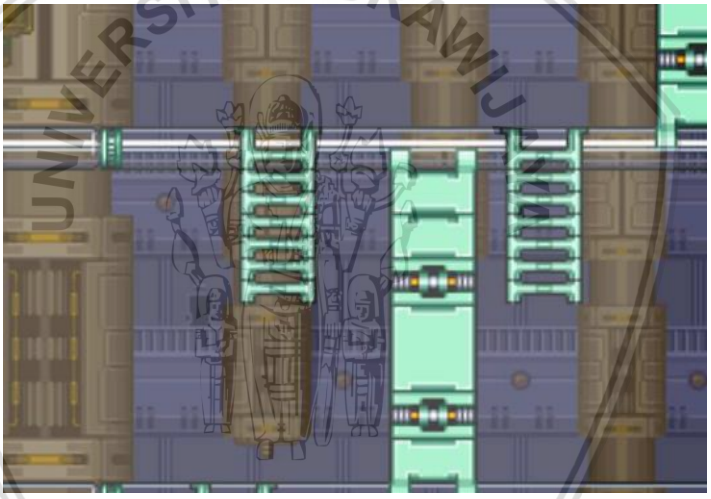
3. *Obstacle*

Obstacle adalah hal utama yang menjadi tantangan di dalam *platform game*. Obstacle adalah obyek apapun yang dapat menimbulkan *damage* atau kerusakan ke *avatar*. Dapat dikatakan contoh dalam hal ini adalah musuh berjalan yang telah di program seperti *bullet* atau peluru. Gambar 2.8 merupakan contoh yang terdapat pada permainan *Sonic the Hedgehog* dimana karakter terkena paku (*spike*).

4. *Movement aids*

Obyek *movement aids* membantu pemain (*player*) melewati *level* dengan cara selain berlari dan juga melompat. Contoh dari *movement aids* dapat berupa tangga, tali, *spring* atau per, bahkan sebuah trampoline seperti pada Gambar 2.9 pada permaianan *Mega Man X2*. Bentuk lainnya adalah *reward* yaitu *power-up* untuk *avatar* atau nyawa tambahan.

Gambar 2.8 Obyek paku (*Spike*) pada *Sonic the Hedgehog*
Sumber : Gillian Smith et al. (2008)



Gambar 2.9 *Movement aids* pada permainan *Mega Man X2*
Sumber : Gillian Smith et al. (2008)

5. *Collectible items*

Semua game ber*genre platformer* memiliki sistem *reward*, dan hampir selalu bentuknya berupa *collectible item* atau benda yang didapatkan. *Collectible item* merupakan obyek apapun di dalam sebuah *level* yang menyediakan *reward* berupa koin, cincin, *power-up*, poin, hingga senjata. Pada Gambar 2.10 sekumpulan pisang yang membentuk tanda panah memberitahukan bagaimana mengakses area tersembunyi pada Donkey Kong. Sistem penghargaan (*reward*) juga penting untuk alasan tertentu, sering kali disiapkan untuk memandu melewati *level* (Bleszinski, 2000).

Gambar 2.10 Pisang pada *Donkey Kong*
Sumber : Gillian Smith et al. (2008)

6. *Triggers*

Triggers atau pemicu merupakan obyek interaktif yang mana avatar dapat menggunakannya sebagai pengubah keadaan suatu level, atau bahkan merubah *rules* dari sebuah *physics* atau fisika permainan. Gambar 2.11 merupakan contoh pemicu dalam membangunkan kotak pada permainan *Super Mario World*.

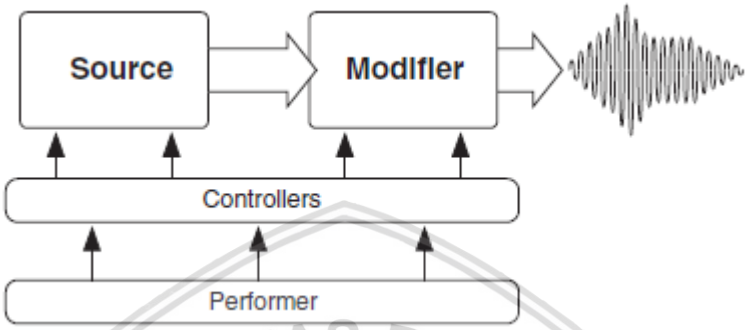
Gambar 2.11 *Scene* pada permainan *Super Mario World*
Sumber : Paper Gillian Smith et al. (2008)

2.3 Suara *Synthesis*

Pembahasan *audio* difokuskan pada *digital audio* yang merupakan representasi dari suara di dalam varietas format digital (Hodgson, 2010). Suara *synthesis* merupakan proses produksi dari suara elektronik. Dalam hal ini dapat mempergunakan kembali suara yang telah ada dengan memprosesnya, atau dapat menghasilkan suara secara eletronik maupun mekanik (Russ, 2009).

Pada Gambar 2.7 *Perfomer* atau seorang pengguna memainkan *instrumen controller* dalam hal ini adalah alat musik, dapat berupa *keyboard* untuk mengubah parameter sebuah *source* dan *modifier* yang merupakan bagian dari

pengaturan, nantinya *output* dari ubahan tersebut menghasilkan suara yang diinginkan. Dengan menggunakan matematika, fisika, atau bahkan hal yang berkaitan dalam dunia biologi, kita dapat mencampurkan seni dan sains ke dalam kemampuan bermusik. Suatu suara akan digabungkan ke unsur-unsur suara yang telah dipilih lainnya menggunakan *sound synthesizer*. *Synthesizer* adalah alat instrumen elektronik yang dapat memproduksi rentang bersuara dari jenis suara yang berbeda. Dalam kata lain *synthesizer* untuk memproduksi suara *synthesis*.



Gambar 2.12 Cara kerja untuk menghasilkan audio synthesis.
Sumber : Martin Russ (2009)

2.4 Efek Suara dan Musik Latar

Efek suara atau *sound effect* (*SFX*) memiliki peran penting selain dari animasi dalam pembuatan sebuah permainan *video*, yang mana dapat membuat pemain merasakan ikut terjun ke dalam dunia permainan (Wei dan Zheng, 2011). Efek suara didefinisikan sebagai suara yang muncul secara singkat, langsung, dan bertahap di dalam proses dari permainan. Sedangkan musik latar atau *Background Music* (*BGM*) dimaksudkan untuk didengarkan secara pasif. Musik latar juga tidak difokuskan untuk didengarkan oleh pendengar atau pemain. Musik yang dimainkan juga memiliki *volume* suara yang kecil.

Hasil survei memperlihatkan kebanyakan dari tiga puluh pemain merasa bahwa musik latar dan efek suara menambah rasa akan ikut terjun ke dalam permainan (*immersion*). Pemain, pengembang, dan pengulas permainan mengutarakan bahwa *immersion* adalah elemen penting dalam mendefinisikan apa yang membuat permainan menjadi diterima (Gormanley, 2013).

Pada Gambar 2.13 merupakan acuan dalam memilih tangga nada untuk pembuatan musik latar dan efek suara. Informasi frekuensi yang diambil dan menjadi acuan dalam penelitian kali ini merupakan hasil penelitian dari *Michigan Technological University* di Amerika. Semua nada memiliki frekuensi yang berbeda untuk membentuk suara tangga nada tersebut. Nada yang dipakai dalam penelitian kali ini berskala hingga tiga oktaf atau tiga tingkat ketinggian suara.

C_4	261.63	131.87
$C^{\#}_4/D^{b}_4$	277.18	124.47
D_4	293.66	117.48
$D^{\#}_4/E^{b}_4$	311.13	110.89
E_4	329.63	104.66
F_4	349.23	98.79
$F^{\#}_4/G^{b}_4$	369.99	93.24
G_4	392.00	88.01
$G^{\#}_4/A^{b}_4$	415.30	83.07
A_4	440.00	78.41
$A^{\#}_4/B^{b}_4$	466.16	74.01
B_4	493.88	69.85
C_5	523.25	65.93
$C^{\#}_5/D^{b}_5$	554.37	62.23
D_5	587.33	58.74
$D^{\#}_5/E^{b}_5$	622.25	55.44
E_5	659.25	52.33
F_5	698.46	49.39
$F^{\#}_5/G^{b}_5$	739.99	46.62
G_5	783.99	44.01
$G^{\#}_5/A^{b}_5$	830.61	41.54
A_5	880.00	39.20
$A^{\#}_5/B^{b}_5$	932.33	37.00
B_5	987.77	34.93
C_6	1046.50	32.97

Gambar 2.13 Frekuensi Nada dalam *Hertz* (Hz)
Sumber : Michigan Techonological University (2018)

2.5 *Unity Engine*

Unity merupakan *platform* raksasa khusus pengembangan game. Unity digunakan untuk membangun *game* tiga dimensi dan dua dimensi dengan kualitas tinggi, meluncurkan *game* tersebut melalui berbagai *platform* seperti *mobile*, *desktop*, *Virtual Reality* / *Augmented Reality*, konsol, atau sebuah *web*, dan dapat terkoneksi dengan para pemain yang antusias dan konsumen. Teknologi Unity menawarkan sebuah *platform* untuk menciptakan keindahan dan hal yang menyenangkan terkait aplikasi permainan dalam bentuk dua dimensi, tiga dimensi, *Virtual Reality*, dan *Augmented Reality*.

2.6 *Audio* Dinamis

Audio dinamis dapat didefinisikan ke dalam dua bentuk. Ketika permainan ingin bereaksi terhadap pemain, tidak hanya dengan visualisasi tetapi juga melalui *audio* yang dapat bereaksi dan berubah untuk menyediakan pengalaman nyata yang imersif. Bentuk *audio* dinamis ini dinamakan interaktif (*interactive*), yang dipicu oleh aksi pemain. Suara permainan yang harus merespon kondisi berlangsungnya sebuah permainan, yang mana dinamakan adaptif (*adaptive*) *audio* (Peerdeman, 2010). Terdapat beberapa teknologi pengembangan untuk mendapatkan *audio* jenis ini. Salah satunya memanfaatkan *Pure Data* yang menghasilkan prosedural *audio* yang merupakan jenis *audio* dinamis. Dari kedua definisi diatas merupakan kelebihan *audio* dinamis dibandingkan dengan *audio* statis (*static*), yang mana sifat dari statis adalah music diam atau tanpa reaksi.

BAB 3 METODOLOGI

Metodologi pada penelitian ini menggunakan tata cara tertentu dalam menyajikan pembahasan *Pure Data*. Terdapat beberapa tahapan dalam menyelesaikan penelitian. Tahap pertama yang dilakukan oleh peneliti adalah studi literatur. Langkah selanjutnya adalah perancangan. Kemudian, peneliti akan melakukan implementasi. Tahap selanjutnya adalah pengujian hasil implementasi dan dilanjutkan ke dalam pengujian. Gambar 3.1 adalah rancangan diagram yang menjelaskan tahap-tahap penelitian yang dilakukan.

Gambar 3.1 Diagram Metodologi Penelitian

3.1 Studi Literatur

Dalam implementasi penelitian kali ini, perlu diadakan studi literatur. Literatur digunakan sebagai teori penguat dan landasan dasar. Pada teori pendukung didapatkan dari jurnal, buku dan internet. Literatur yang akan digunakan di dalam penelitian ini adalah:

1. *Pure Data*

   Pure Data akan dipelajari sebagai landasan utama teknologi yang diterapkan di dalam penelitian. Implementasi yang digunakan ada dua yaitu, musik latar (*background music*) dan efek suara (*sound effect*). *Pure Data* diimplementasikan secara langsung dengan memanfaatkan kerangka kerja *Heavy*.

2. Suara *Synthesis*

   Suara *synthesis* digunakan untuk jenis suara yang akan diterapkan pada penelitian. Seringkali dalam menghasilkan jenis suara, *Pure Data* menghasilkan suara *synthesis*. Oleh karena itu, literatur terkait suara *synthesis* menjadi bahan studi pada penelitian.

3. *Unity Engine*

   Unity merupakan *game engine* untuk mengembangkan permainan. *Unity* digunakan Untuk pembuatan permainan *platfomer* yang akan diterapkan pada penelitian ini. *Pure Data* dapat diimplementasikan ke dalam *Unity* dengan memanfaatkan kerangka kerja *Heavy*.

4. Permainan *Platformer*

   Permainan *Platformer* akan dipelajari untuk menunjang penelitian dalam mengimplementasikan *Pure Data*. Permainan *Platformer* memiliki beberapa komponen yang digunakan sebagai desain sebuah *level*. Komponen *platformer* terdiri dari *avatar*, landasan (*platform*), rintangan (*obstacles*), alat bantu gerak (*movement aids*), benda koleksi (*collectible item*), dan pemicu (*trigger*) (Smith et al., 2008).

3.2 Perancangan *Pure Data*

Perancangan dilakukan untuk merancang tahapan pertama pada metode penelitian. Pada subbab ini menjelaskan bagaimana perancangan diimplementasikan dan bagian mana pada studi literatur yang diterapkan untuk *Real-time Audio Synthesis*. Perancangan pada *Platformer game* dilakukan dengan maksud untuk mendapatkan rancangan *Pure Data patch* atau bentuk dokumen sebuah *Pure Data*, sedangkan perancangan pada *Pure data* mempunyai tujuan untuk menjadi dasar efek suara dan musik latar.

3.3 Implementasi *Pure Data* pada *Unity*

Implementasi mengikuti perancangan yang telah dilakukan. Implementasi akan dilakukan di dalam *Unity* dengan memanfaatkan *Pure Data patch* sebagai *sound engine*. Implementasi harus sesuai dengan hasil perancangan. Selanjutnya akan dilakukan sesuai dengan metode penelitian. Implementasi *Pure Data ke Unity* memanfaatkan *Heavy* sebagai penghasil skrip *audio*.

3.4 Pengujian Hasil *Real-time Audio Synthesis* di *Platformer Game*

Bagian ini menjelaskan pengujian terhadap hasil *Real-time Audio Synthesis*. Pengujian menggunakan beberapa kasus uji diantara lainnya adalah:

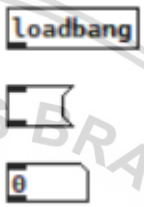
1. Melakukan pengujian unit untuk menguji algoritma dari suatu skrip permainan, menentukan *valid* atau tidak *valid* nya suatu algoritma program pada penelitian.

2. Melakukan pengujian validasi. Terdapat beberapa kasus uji yang mencakup keseluruhan dari implementasi *Pure Data* ke dalam permainan. Contoh dari kasus uji ini adalah menguji ketepatan *audio* apakah berjalan dan berubah secara dinamis sesuai dengan kondisi yang telah ditetapkan di perancangan.
3. Melakukan perbandingan. Hasil dari *audio mp3* dan hasil dari implementasi *Pure Data* dibandingkan satu sama lain.

BAB 4 PERANCANGAN

4.1 *Pure Data Patch*

Pada tahap ini akan dijelaskan bagaimana perancangan semua aset suara. Perancangan dilakukan dengan memanfaatkan bahasa pemrograman visual *Pure Data*. Dokumen yang dihasilkan dari bahasa pemrograman tersebut adalah *Pure Data patch* atau *patch*. Dari perancangan ini menghasilkan dua jenis *patch* yaitu efek suara dan musik latar. Musik latar terbagi menjadi dua yaitu suara angin dan menu utama. Semua jenis suara hasil dari perancangan akan diimplementasikan ke dalam *Unity* melalui bantuan kerangka kerja *Heavy* yaitu menjembatani antara *Pure Data patch* ke dalam *Unity*.



Gambar 4.1 Contoh jenis kotak pada *Pure Data patch*

Di dalam *Pure data patch* atau *patch* terdapat obyek-obyek yang dapat dihubungkan satu sama lain, dan memiliki fungsi yang berbeda-beda dalam pemogramannya. Gambar 4.1 merupakan beberapa contoh kotak atau *box*. Dimulai dari atas yaitu kotak *object*, *message*, dan *number*.

4.1.1 *Patch* Efek Suara

Pada permainan *platformer* di dalam penelitian ini efek suara akan hadir pada *scene* permainan. Yaitu pada sesi ketika karakter mengambil benda koleksi atau *collectible item*. Setiap benda koleksi yang didapatkan akan menghasilkan efek suara yang berbeda. Perancangan efek suara akan dijelaskan beserta *card Pure Data patch* atau referensi kartu-kartu yang dipakai dalam perancangan. Pada Tabel 4.1 merupakan penjelasan mengenai kartu-kartu yang digunakan untuk merancang *Pure Data patch* efek suara.

Tabel 4.1 Kartu *Pure Data patch* pada Efek Suara

Nama Kartu	Jenis Kotak	Deskripsi	Keterangan
select	*Object*	Membandingkan angka atau simbol	Pada *patch* dari efek suara, beberapa kartu *select* bertugas memilih angka yang sesuai dengan masukan
loadbang	*Object*	Mengirim pesan "*bang*" pada saat memuat	Pada *patch* dari efek suara, beberapa kartu *loadbang* bertujuan memuat *message*

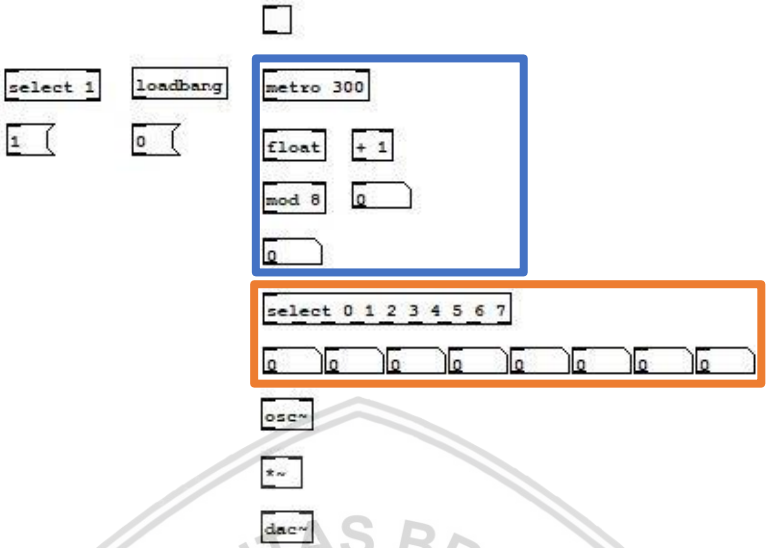
			yang berpengaruh pada kekonsistenan program
metro	*Object*	Mengirim serangkaian "*bang*" pada waktu yang ditentukan	Pada *patch* dari efek suara, beberapa kartu *metro* bertugas menjadi sebuah mesin metronom untuk kecepatan keluaran suara yang dihasilkan
float	*Object*	Penyimpan angka dalam hal ini *floating number*	Pada *patch* dari efek suara, beberapa kartu *float* menyimpan angka hasil dari perhitungan sebelumnya, ditujukan untuk pengulangan
+	*Object*	Operator perhitungan aritmetika	Pada *patch* dari efek suara, merupakan operasi pertambahan, ditujukan untuk pengulangan
mod	*Object*	Modulus merupakan sisa pembagian dari suatu bilangan yang lain	Pada *patch* dari efek suara, beberapa kartu *mod* memiliki tujuan operasi aritmetika, ditujukan untuk pengulangan
osc~	*Object*	*Cosine wave oscillator*	Pada *patch* dari efek suara, kartu *osc~* merupakan obyek yang memiliki keluaran berupa *cosine wave*
*~	*Object*	Operator di dalam *audio signal*	Pada *patch* dari efek suara, beberapa kartu *~ berperan sebagai operator *audio signal* dalam hal ini perkalian gelombang
dac~	*Object*	Penyedia keluaran di *Pure Data*	Pada *patch* dari efek suara, kartu *dac~* berperan sebagai keluaran *speaker* dari hasil obyek-obyek yang telah dibentuk

delay	*Object*	Memanggil kembali setelah waktu keterlambatan atau *delay*	Pada *patch* dari efek suara, kartu *delay* berperan sebagai waktu keterlambatan dalam pergantian suara
receive	*Object*	Cara lain mendapatkan pesan, dapat terhubung maupun tidak	Pada *patch* dari efek suara, beberapa kartu *receive* menyalurkan dari dalam *Unity*
unpack	*Object*	Memisahkan pesan menjadi atom-atom	Pada patch dari efek suara, unpack mengambil data frekuensi dan memisahkan ke dalam setiap kotak *number* atau angka
-	*Message*	Kotak *message* atau pesan adalah kotak teks yang dapat membuat sebuah pesan	Pada *patch* dari efek suara, kotak pesan menyimpan pesan yang nantinya akan dipakai oleh kotak obyek lainnya
-	*Number*	kotak *number* atau angka memperbolehkan angka ditampilkan atau memasukan angka	Pada *patch* dari efek suara, kotak angka menampilkan angka yang telah dipilih dan angka-angka tersebut digunakan sesuai dengan tujuannya

Tahapan pertama dengan memulai membuat sebuah *Pure Data patch* pada Gambar 4.2. Diawali dengan pembuatan delapan titik *sequencer* sederhana yang mana *sequencer* merupakan alat perangkat keras atau perangkat lunak yang dapat merekam, menerima atau mengirim, mengolah bunyi digital dari suatu instrumen musik elektronik (Widodo, 2006) yang berada pada penjelas warna oranye. Terdapat 8 kotak *number* dan kotak *object* bernama "*select*". Delapan kotak *number* ini berfungsi sebagai penyimpan frekuensi nada. Frekuensi nada didapatkan melalui referensi pada Gambar 2.13. Kotak *object* bernama "*select*" memiliki delapan argumen mulai dari angka 0 sampai 7. Delapan argumen akan dijalankan berurutan sesuai dengan tempo yang telah diprogram.

Selanjutnya pada penjelas warna biru terdapat kotak *object* bernama "*metro*", "*float*","+", dan "*mod*" dan memiliki argumen masing-masing. Kotak "*metro*" dengan argumen 300 adalah mesin metronom atau pengatur tempo yang memiliki kecepatan 300ms atau 300 milidetik. Jika disatukan dengan kotak "*float*", "+", dan "*mod*" yang merupakan operasi matematika maka terjadi penambahan 1 yang berulang yang ditampilkan pada kotak *number* dibawahnya. Angka yang ditampilkan akan dimodulus sehingga hasil dari habis dibagi 8 menjadi acuan kotak

"*select*" berjalan berurutan sesuai dengan tempo dan operasi matematika dipenjelas warna oranye.



Gambar 4.2 Perancangan bagian ke-1 *patch* Efek Suara

Gambar 4.3 Perancangan bagian ke-2 *patch* Efek Suara

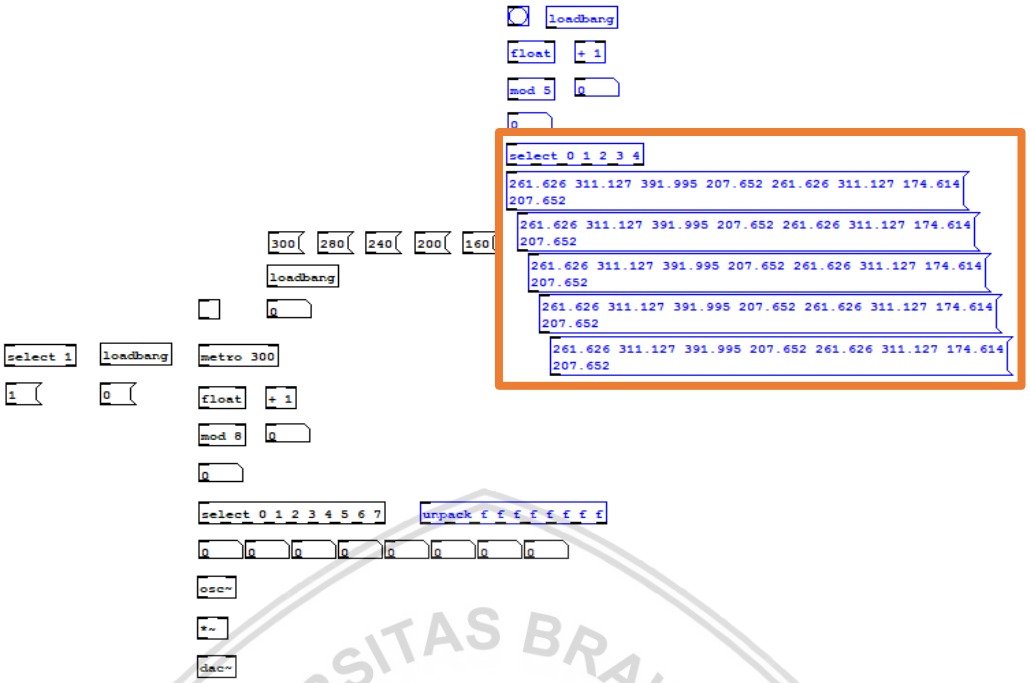
Tahap kedua pada Gambar 4.3 penjelas warna oranye, memastikan pembuatan kotak *message* untuk pilihan tempo yang tersedia. Pilihan tempo yang ada didapatkan melalui percobaan oleh peneliti. Terdapat kotak *object* "*loadbang*" yang berfungsi untuk memuat pilihan tempo. Ketika program dijalankan pilihan tempo tetap memuat angka yang sudah terpilih sebelumnya dan ditampilkan pada kotak *number* dibawahnya.

Tahap Ketiga pada Gambar 4.4 penjelas warna oranye, terdapat beberapa kotak *message* yang merupakan daftar dari kumpulan frekuensi. Daftar ini merupakan kumpulan frekuensi yang telah disesuaikan mengikuti referensi pada Gambar 2.13. Ukuran frekuensinya sesuai dengan nada do-re-mi-fa-so-la-si-do untuk membuat sebuah *chord* atau nada alunan musik. Sebuah *chord* dari kotak *message* tersebut dibuat oleh peneliti dengan menggunakan referensi alat musik *keyboard*. Kotak *object* bernama "*select*" mempunyai fungsi yang sama dengan tahap sebelumnya yaitu dijalankan berurutan sesuai dengan tempo yang telah diprogram. Hasil rancangan pada Gambar 4.4 total terdapat lima pilihan *chord* atau nada alunan musik yang nantinya berperan sebagai efek suara disaat pemain mendapatkan benda koleksi atau *collectible item*.

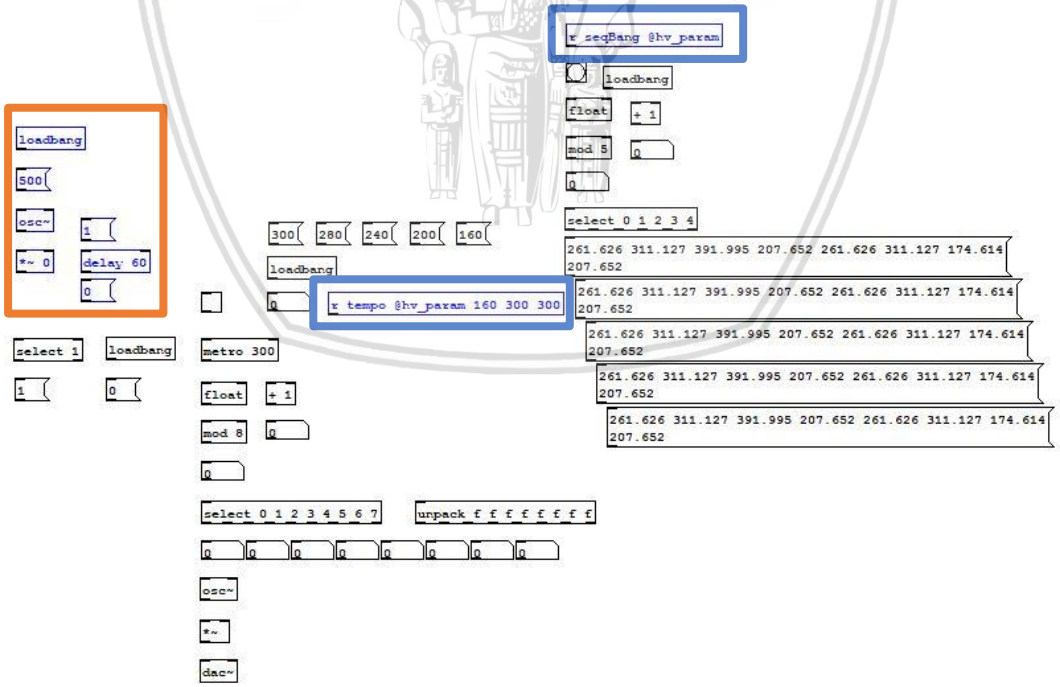
Tahap keempat pada Gambar 4.5 penjelas warna oranye adalah pembuatan suara *beep* singkat sebagai jeda, yang mana akan bereaksi ketika perpindahan *chord* disaat mendapatkan benda koleksi (*collectible item*). Terdapat kotak *object* bernama "*loadbang*", "*osc~*", "*~", dan "*delay*". *Cosine wave oscillator* atau "*osc~*" merupakan pengubah input untuk menjadi sinyal *audio*, sedangkan "*~" merupakan operasi perkalian yang ditujukan untuk melipat gandakan sinyal. Kotak *object "delay*" ditujukan untuk memberi efek *delay* atau terlambat pada suara *beep*.

Selanjutnya pada penjelas warna biru, memastikan untuk membuat kotak *object* "*receive*" atau "*r*" beserta argumen masing-masing. Terdapat argumen *tempo* dan *seqBang*, diakhiri dengan *@hv_param*. Tujuan dari dua kotak *object* ini untuk kepentingan di dalam memprogram hasil suara pada *Unity*. Argumen *@hv_param* berfungsi untuk mengakses fungsionalitas dari kerangka kerja *Heavy*. Argumen *tempo* merupakan inisialisai merujuk pada fungsionalitas *tempo* untuk mengatur kecepatan dari musik, sedangkan argumen *seqBang* merujuk pada fungsionalitas perubahan *chord* atau nada alunan musik.

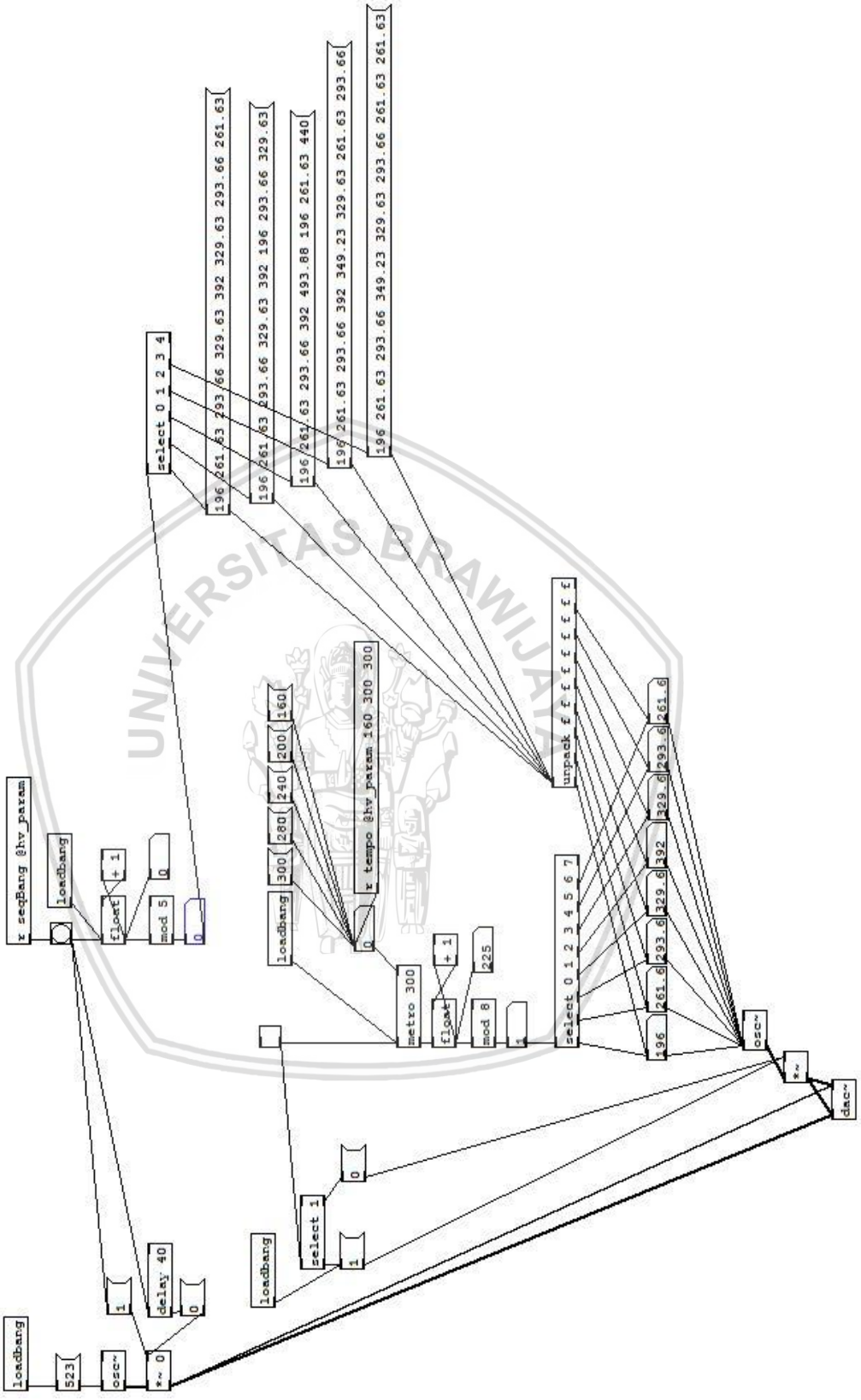
Tahap kelima atau terakhir pada Gambar 4.6, merupakan hasil akhir dari rancangan *patch* efek suara. Setelah hasil rancangan diselesaikan, *patch* disambungkan sesuai dengan tujuan program sehingga fungsi-fungsi setiap obyek yang dibuat dapat bekerja. Hasil akhir dari rancangan siap untuk diimplementasikan melalui perangkat kerangka kerja, yaitu alat bantu *Heavy*. Setelah semua implementasi diselesaikan, hasil akhir dari kerangka kerja *Heavy* akan menjadi aset suara dan dapat digunakan di dalam permainan. *Audio* inilah yang digunakan dalam pembangunan permainan *platformer.*



Gambar 4.4 Perancangan bagian ke-3 *patch* Efek Suara



Gambar 4.5 Perancangan bagian ke-4 *patch* Efek Suara



Gambar 4.6 Perancangan bagian ke-5 *patch* Efek Suara

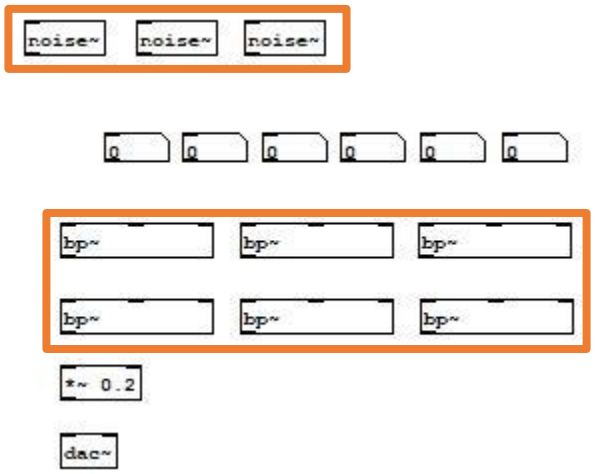
4.1.2 *Patch* Musik Latar

Musik latar (*background music*) di dalam permainan yang dibuat pada penelitian kali ini adalah suara angin dan menu utama. Pada saat game dimainkan musik latar suara angin akan terdengar tidak terlalu keras *volume* nya dan berjalan hingga permainan telah selesai, sedangkan musik latar menu dimainkan ketika pemain berada di menu utama. Pada Tabel 4.2 menjelaskan bagian-bagian obyek yang digunakan pada perancangan musik latar (*background music*) untuk membuat suara angin.

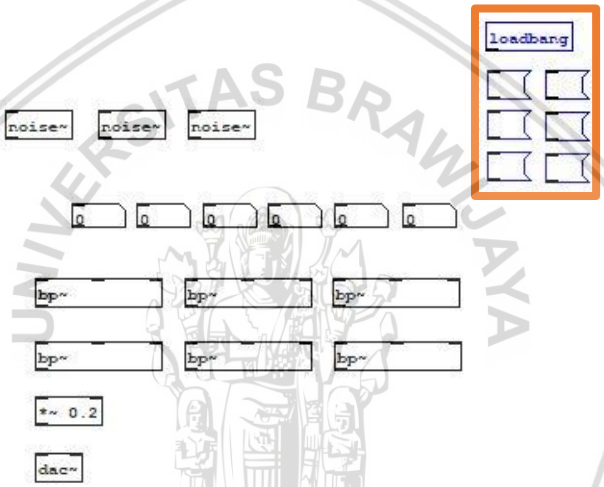
Tabel 4.2 Kartu *Pure Data patch* pada Musik Latar Suara Angin

Nama *Kartu*	Jenis Kartu	Deskripsi	Keterangan
noise~	*Object*	Pendistribusian suara *white noise*	Pada *patch* dari musik latar suara angin, beberapa kartu *noise* berperan mengeluarkan suara *noise*
bp~	*Object*	penyaring *BANDPASS*	Pada *patch* dari musik latar suara angin, beberapa kartu
*~	*Object*	Operator di dalam *audio signal*	Pada *patch* dari musik latar suara angin, kartu *~ berperan sebagai operator *audio signal* dalam hal ini perkalian
dac~	*Object*	*Audio input* dan *output*	Pada *patch* dari musik latar suara angin, kartu *dac~* berperan sebagai keluaran dari hasil obyek-obyek yang telah dibentuk
loadbang	*Object*	Mengirim pesan "*bang*" pada saat memuat	Pada *patch* dari musik latar suara angin, kartu *loadbang* bertujuan memuat *message* yang berpengaruh pada kekonsistenan program

Tahap pertama pada Gambar 4.7 penjelas warna oranye, membuat tiga buah *noise generator* yaitu kotak *object* bernama "*noise~*" dan *bandpass filter* yaitu kotak *object* bernama "*bp~*". Dengan menggabungkan keduanya dapat lebih mengontrol 'faktor-Q' berkaitan dengan hasil frekuensi. *Noise generator* berfungsi menghasilkan suara *noise*, sedangkan *bandpass filter* untuk menyaring suara *noise*. Terdapat enam buah *bandpass filter* bertujuan menghasilkan efek kedalaman suara.



Gambar 4.7 Perancangan bagian ke-1 *patch* Musik Latar Suara Angin

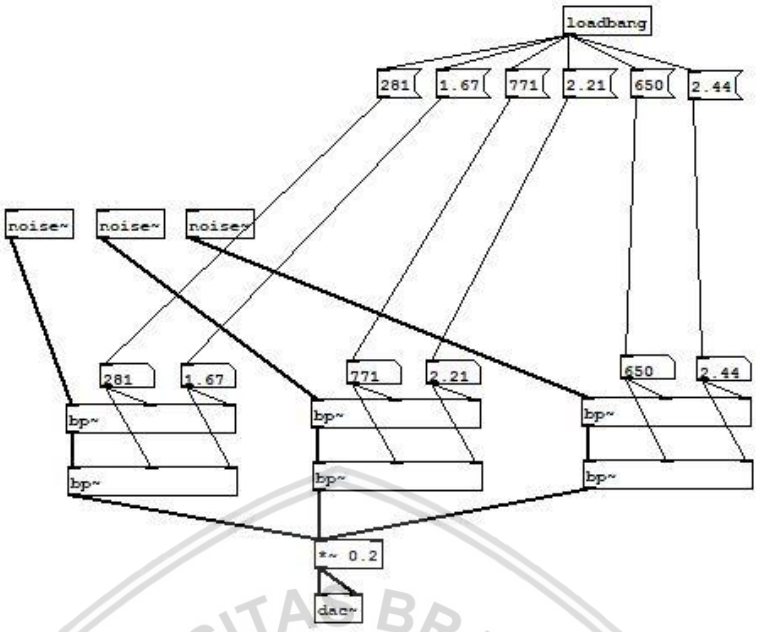


Gambar 4.8 Perancangan bagian ke-2 *patch* Musik Latar Suara Angin

Penggunaan perkalian sinyal atau "*~" untuk menggandakan sinyal. Mengkalikan sinyal ke 0.2 bermaksud untuk mendapatkan zona aman suara yang sehingga frekuensi dapat dimainkan sesuai dengan nilai yang digunakan. Selanjutnya memastikan bahwa suara berada pada tingkatan yang benar dengan mengetahui keluaran melalui kotak *object* "*dac~*".

Tahap Kedua pada Gambar 4.8 penjelas warna oranye, membuat kotak *message* untuk menetapkan nilai awal atau *default* pada setiap kotak *number*. Kotak *message* akan diisi dengan beberapa frekuensi, sedangkan sebuah "*loadbang*" akan memuat pesan dari kotak *message. Patch* akan bekerja sesuai dengan program yang diinginkan ketika diimplementasikan ke dalam *Unity*.

Tahap ketiga atau terakhir pada Gambar 4.9, dalam perancangan suara angin sebagai musik latar (*background music*) adalah mengisi frekuensi yang sesuai dan meletakan nilai awal frekuensi ke dalam kotak *message*. Gambar 4.9 merupakan hasil akhir rancangan *patch* suara angin sebagai musik latar (*background music*).



Gambar 4.9 Perancangan bagian ke-3 *patch* Musik Latar Suara Angin

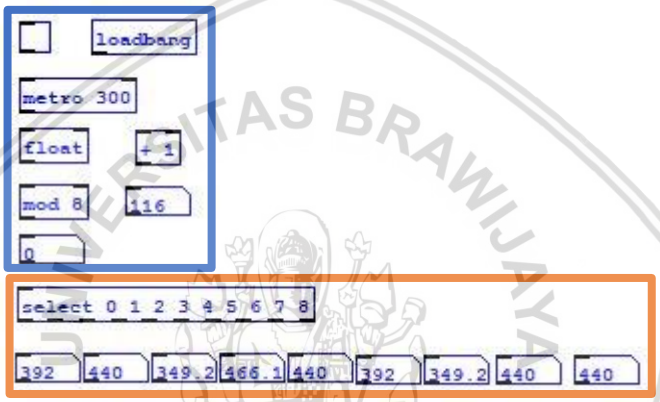
Pada Tabel 4.3 menjelaskan kartu apa saja yang digunakan untuk referensi di dalam perancangan musik latar menu utama. Musik latar menu utama akan dimainkan ketika pemain berada di dalam menu utama permainan. Musik latar menu utama memiliki perancangan yang hampir sama dengan perancangan efek suara.

Tabel 4.3 Kartu *Pure Data patch* pada Musik Latar Menu Utama

Nama Kartu	Jenis Kartu	Deskripsi	Keterangan
select	*Object*	Membandingkan angka atau simbol	Pada *patch* dari musik latar menu utama, beberapa kartu *select* bertugas memilih angka yang sesuai dengan masukan
loadbang	*Object*	Mengirim pesan "*bang*" pada saat memuat	Pada *patch* dari musik latar menu utama, beberapa kartu *loadbang* bertujuan memuat *message* yang berpengaruh pada kekonsistenan program
metro	*Object*	Mengirim serangkaian "*bang*" pada waktu yang ditentukan	Pada *patch* dari musik latar menu utama, beberapa kartu *metro* bertugas menjadi sebuah mesin metronom untuk kecepatan

			keluaran suara yang dihasilkan
float	*Object*	Penyimpan angka dalam hal ini *floating number*	Pada *patch* dari musik latar menu utama, beberapa kartu *float* menyimpan angka hasil dari perhitungan sebelumnya, ditujukan untuk pengulangan
+	*Object*	Operator perhitungan aritmetika	Pada *patch* dari musik latar menu utama, merupakan operasi pertambahan, ditujukan untuk pengulangan
mod	*Object*	Modulus merupakan sisa pembagian dari suatu bilangan yang lain	Pada *patch* dari musik latar menu utama, beberapa kartu *mod* memiliki tujuan operasi aritmetika, ditujukan untuk pengulangan
osc~	*Object*	*Cosine wave oscillator*	Pada *patch* dari musik latar menu utama, kartu *osc~* merupakan obyek yang memiliki keluaran berupa *cosine wave*
*~	*Object*	Operator di dalam *audio signal*	Pada *patch* dari musik latar menu utama, beberapa kartu *~ berperan sebagai operator *audio signal* dalam hal ini perkalian gelombang
dac~	*Object*	Penyedia keluaran di *Pure Data*	Pada *patch* dari musik latar menu utama, kartu *dac~* berperan sebagai keluaran *speaker* dari hasil obyek-obyek yang telah dibentuk
unpack	*Object*	Memisahkan pesan menjadi atom-atom	Pada patch dari musik latar menu utama, unpack mengambil data frekuensi dan memisahkan ke dalam setiap kotak *number*

-	*Message*	Kotak *message* atau pesan adalah kotak teks yang dapat membuat sebuah pesan	Pada *patch* dari musik latar menu utama, kotak pesan menyimpan pesan yang nantinya akan dipakai oleh kotak obyek lainnya
-	*Number*	kotak *number* atau angka memperbolehkan angka ditampilkan atau memasukan angka	Pada *patch* dari musik latar menu utama, kotak angka menampilkan angka yang telah dipilih dan angka-angka tersebut digunakan sesuai dengan tujuannya



Gambar 4.10 Perancangan bagian ke-1 *patch* Musik Latar Menu Utama

Tahap pertama pada Gambar 4.10 penjelas warna oranye, merupakan tahap awal yang dilakukan yaitu membuat sembilan titik sequencer dengan memanfaatkan kotak *object* bernama "*select*", dan kotak *number*. Seperti pembahasan sebelumnya pada efek suara, "*select*" berfungsi untuk menjalankan kesembilan argumen mulai dari 0 sampai 8 secara berurutan. Kotak *number* akan menampilkan frekuensi setiap nada yang dipilih dan membuat alunan musik sesuai dengan nada yang disimpan.

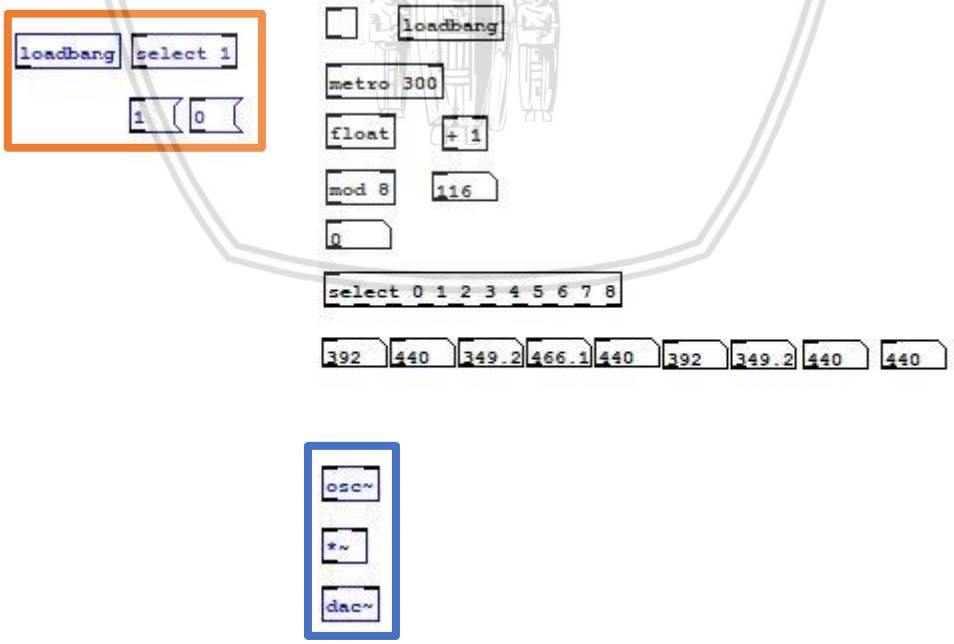
Selanjutnya pada penjelas warna biru, pembuatan mesin metronom sama dengan perancangan efek suara, yaitu sebagai pengatur *tempo* yang diperlukan untuk kecepatan operasi matematika. Fungsi dari kotak *object* bernama "*float*", "*mod*", dan "+" sebagai operasi matematika. Hasil dari operasi itu ditampilkan dan terhubung dengan argumen "*select*" pada penjelas warna oranye. Kecepatan operasi matematika mempengaruhi kecepatan "*select*" dalam memilih argumen, sehingga alunan musik juga ikut terpengaruh.

Tahap kedua pada Gambar 4.11 penjelas warna oranye, membuat "*loadbang*" dan "*select*" dengan argumen angka 1 atau "*on*" yang ditujukan untuk memuat keseluruhan pergerakan *patch*, sehingga nada akan tetap berjalan sesuai dengan operasi. Untuk dapat membuat keluaran sehingga sinyal dapat dikeluarkan melaui

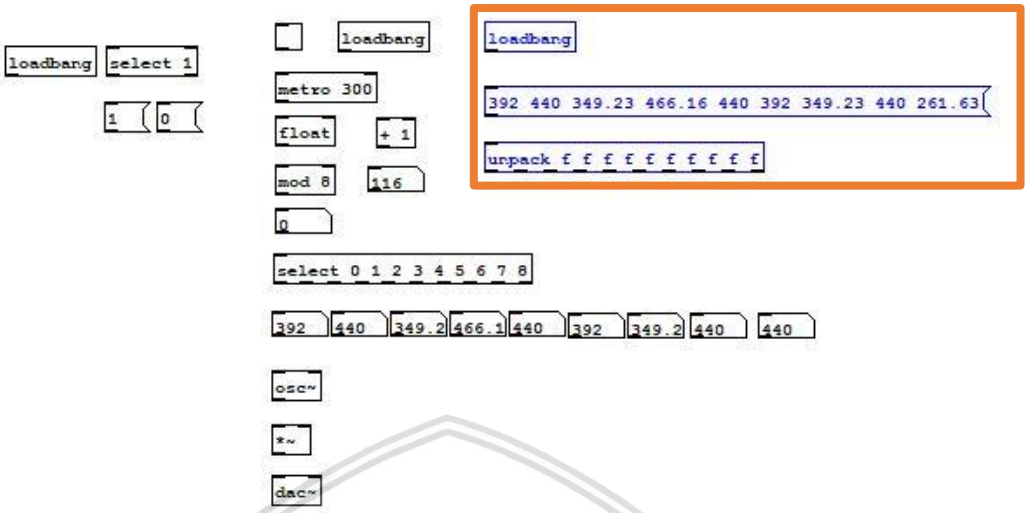
speaker dan mengeluarkan suara yang sesuai dengan program maka dibutuhkan "*dac~*" sebagai *digital audio converter*. Kotak *object* bernama "osc~" bertugas sebagai pengubah input untuk membuat sinyal audio dan selanjutnya sinyal tersebut digandakan dengan memanfaatkan "*~" atau operasi perkalian sinyal, dijelaskan pada penjelas warna biru.

Tahap Ketiga pada Gambar 4.12 penjelas warna oranye, pembuatan musik latar pada kesempatan kali ini hanya membutuhkan satu kotak *message* berisi pesan dengan sembilan frekuensi yang telah dipilih sebelumnya, dan diurutkan sehingga membentuk *chord* atau alunan musik yang ditujukan untuk musik latar pada menu utama. Terdapat kotak *object* bernama "*unpack*" dengan 9 argumen *f* yang bertujuan untuk memisahkan sembilan frekuensi yang terkumpul pada kotak *message*. Sembilan frekuensi tersebut akan disimpan pada kotak *number* dan dipisahkan ke masing-masing kotak sesuai dengan Gambar 4.10 penjelas warna oranye. Jenis *chord* yang dimainkan pada musik latar menu utama hanya satu alunan musik dan diulang secara terus-menerus. Alasan hanya memakai satu jenis *chord* dikarenakan musik latar menu utama adalah jenis suara yang pasif atau suara yang dimainkan hanya pada latar permainan.

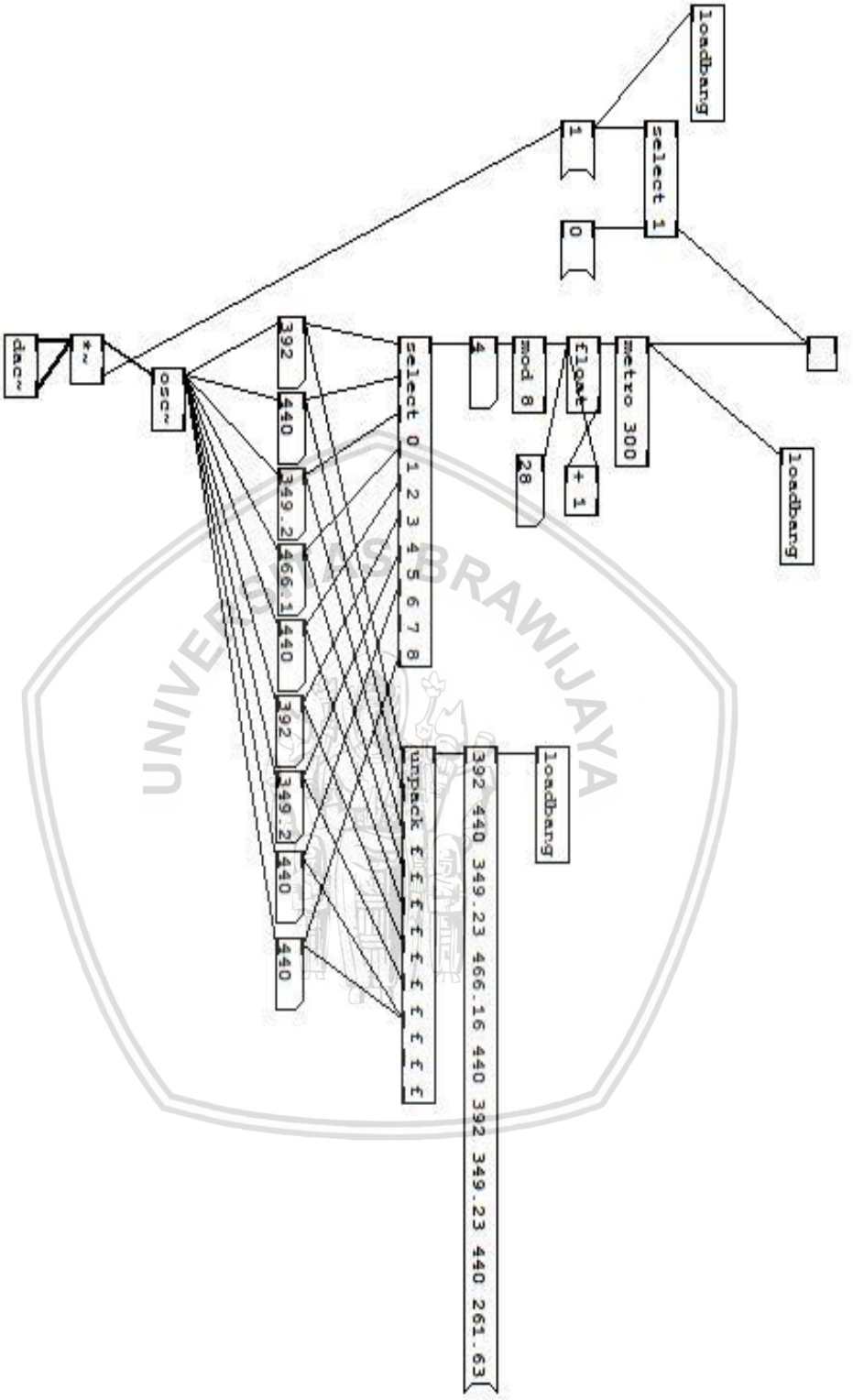
Tahap keempat atau terakhir pada Gambar 4.13, merupakan tahap untuk menyatukan setiap kotak sesuai dengan cara kerja program yang diingikan. Selanjutnya membentuk satu kesatuan *patch* untuk musik latar pada menu utama di dalam permainan *platformer*. *Patch* untuk musik latar menu utama siap untuk diimplementasikan ke dalam *Unity* memanfaatkan kerangka kerja *Heavy* sebagai penghubung diantara keduanya.



Gambar 4.11 Perancangan bagian ke-2 *patch* Musik Latar Menu Utama



Gambar 4.12 Perancangan bagian ke-3 *patch* Musik Latar Menu Utama

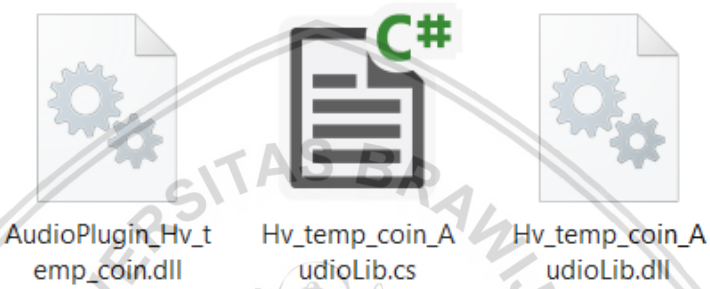


Gambar 4.13 Perancangan bagian ke-3 *patch* Musik Latar Menu Utama

BAB 5 IMPLEMENTASI

5.1 Implementasi *Heavy*

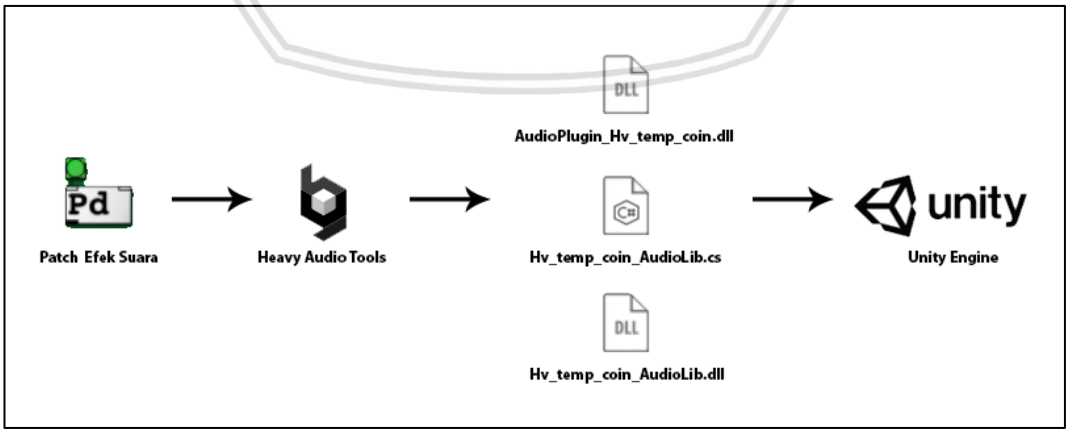
Pada pembahasan kali ini akan dijelaskan implementasi *Heavy* yang merupakan kerangka kerja untuk menghasilkan *audio* berupa skrip dalam Bahasa program C#. Komponen yang berada di dalam skrip tersebut berasal dari perancangan *Pure Data* sebelumnya. Sehingga dapat dikatakan *Heavy* menghubungkan fungsionalitas *Pure Data* ke *Unity*. Hasil yang diperoleh dari *Heavy* selanjutnya akan dimanfaatkan sebagai *plugin audio* pada permainan *platformer* yang telah dibuat.



Gambar 5.1 Berkas-berkas hasil dari kerangka kerja *Heavy*

Terdapat tiga berkas yang dihasilkan melalui kerangka kerja *Heavy*. Salah satu berkas tersebut adalah skrip *audio* yang dapat dimodifikasi sesuai kebutuhan. Jenis berkas lainnya adalah *library* atau sekumpulan kode yang disusun untuk dapat diakses dan digunakan oleh program lain dalam hal ini digunakan dalam penulisan kode program C#. Gambar 5.1 menunjukkan ketiga berkas dari salah satu implementasi yang telah dihasilkan.

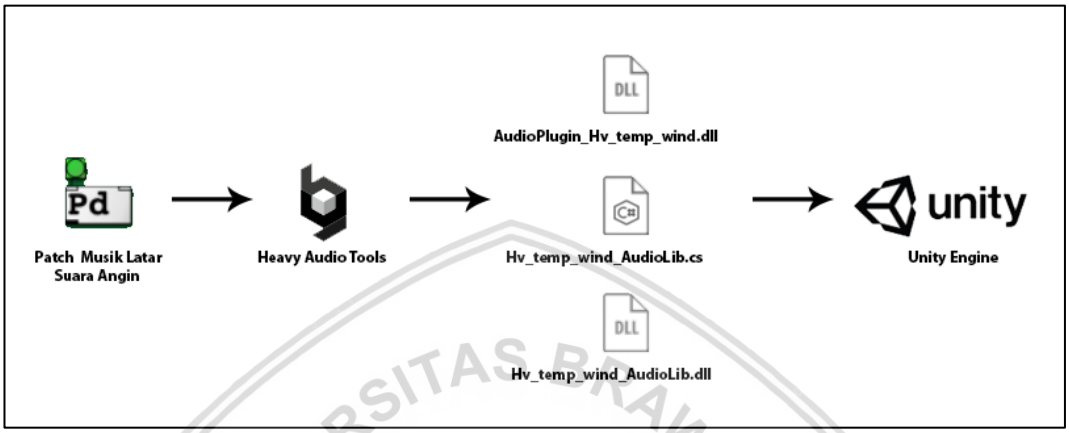
5.1.1 Implementasi Efek Suara



Gambar 5.2 Implementasi Efek Suara

Gambar 5.2 menunjukkan bagaimana proses implementasi dari efek suara. Mulai dari *Pure Data patch* yang merupakan bentuk perancangan *audio*, dilanjutkan dengan implementasi ke dalam *Heavy*. Skrip *audio* efek suara yang terbentuk melalui tahap modifikasi pada sebagian kode dengan tujuan tertentu dan untuk dapat diimplementasikan ke dalam *Unity*.

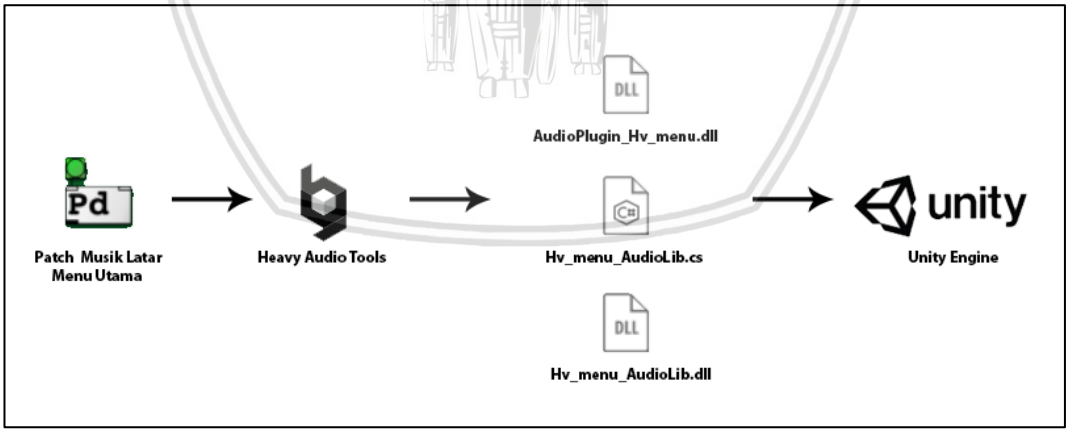
5.1.2 Implementasi Musik Latar Suara Angin



Gambar 5.3 Implementasi Musik Latar Suara Angin

Gambar 5.3 merupakan implementasi dari musik latar suara angin. *Pure Data patch* suara angin juga memanfaatkan *Heavy* sebagai penghasil skrip *audio*. Skrip *audio* musik latar suara angin yang terbentuk diimplementasikan secara langsung ke dalam *Unity*.

5.1.3 Implementasi Musik Latar Menu Utama



Gambar 5.4 Implementasi Musik Latar Menu Utama

Gambar 5.4 adalah proses implementasi dari musik latar menu utama. *Pure Data patch* dari musik latar menu utama dirancang untuk dapat diimplementasikan ke dalam *Unity*. Melalui kerangka kerja *Heavy* menghasilkan skrip permainan yang dapat secara langsung diimplementasikan ke dalam *Unity*.

5.2 Implementasi Skrip *Audio*

Skrip *audio* dapat langsung diimplementasikan ke dalam *Unity* tanpa harus ditulis ulang atau dimodifikasi. Dalam penelitian kali ini, skrip *audio* yang dimodifikasi hanya pada bagian *Hv_temp_coin_AudioLib.cs* yang berada di Gambar 5.2, sedangkan skrip *audio* pada Gambar 5.3 dan 5.4 diimplementasikan secara langsung ke dalam permainan. Modifikasi pada skrip *Hv_temp_coin_AudioLib.cs* bertujuan untuk membuat suatu kondisi pada *method Update()*, agar dapat menjalankan fungsionalitas *Heavy* yaitu merubah *chord* atau nada alunan musik pada Gambar 4.4. *Hv_temp_coin_AudioLib.cs* merupakan skrip *audio* yang dihasilkan dari perancangan *Pure Data patch* pada Gambar 4.5. Kemudian, pada skrip tersebut terdapat modifikasi sesuai dengan Tabel 5.2. Bagian kode program tersebut ditulis di dalam *method Update()*, bertujuan agar kode terpanggil disetiap *frame* atau banyaknya tampilan yang dihasilkan di dalam permainan.

Tabel 5.1 Skrip Permainan *Player_Score.cs*

No	Kode Program
1	`public class Player_Score : MonoBehaviour {`
	`. . .`
10	`public static int count;`
	`. . .`
15	`void Start() {`
	`. . .`
18	`count = 0;`
19	`}`
	`. . .`
36	`void OnTriggerEnter2D(Collider2D collision)`
37	`{`
	`. . .`
44	`if (collision.gameObject.name == "Coin")`
45	`{`
	`. . .`
50	`count = count + 1;`
	`. . .`
53	`}`
55	`}`
	`. . .`
72	`}`

Tabel 5.2 Implementasi *Method Update()* Skrip *Hv_temp_coin_AudioLib.cs*

No	Kode Program
128	`private int tempcount = 0;`
	`. . .`
197	`private void Update() {`
	`. . .`
207	`else if (Player_Score.count > tempcount)`
208	`{`
209	`_context.SendFloatToReceiver((uint)Parameter.Seqbang, seqBang);`
210	`tempcount++;`
211	`}`
	`. . .`
215	`}`

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA

Kode pada Tabel 5.2 memanfaatkan kondisi yang terjadi pada skrip permainan *Player_Score.cs*. Jika variabel *count* yang terdapat pada skrip di Tabel 5.1 lebih besar dari variabel *tempcount* maka efek suara akan berganti sesuai dengan *Pure Data patch* yang dirancang sebelumnya.

Pada Tabel 5.1 merupakan skrip yang bertugas untuk mengatur skor pemain dalam permainan. M*ethod OnTriggerEnter2D* pada skrip baris 36 memiliki fungsi sebagai pengatur kejadian tabrakan. Skrip baris 44 adalah suatu kondisi ketika pemain menabrak obyek permainan bernama "*Coin*" maka variabel global *count* akan bertambah satu. Pada skrip *Player_Score.cs* memiliki hubungan dengan *method Update()* pada Tabel 5.2.

Pada Tabel 5.2 merupakan skrip hasil kerangka kerja *Heavy* implementasi dari efek suara pada Gambar 5.2. Skrip *Hv_temp_coin_AudioLib.cs* berisi fungsionalitas kerangka kerja *Heavy*. Terdapat modifikasi pada *method Update()* yaitu kondisi ketika variabel *count* yang terdapat pada Tabel 5.1 lebih besar dari variabel global *tempcount* maka fungsionalitas *SeqBang* yang bertugas sebagai pengatur pergantian alunan musik akan dijalankan, serta variabel *tempcount* akan bertambah.

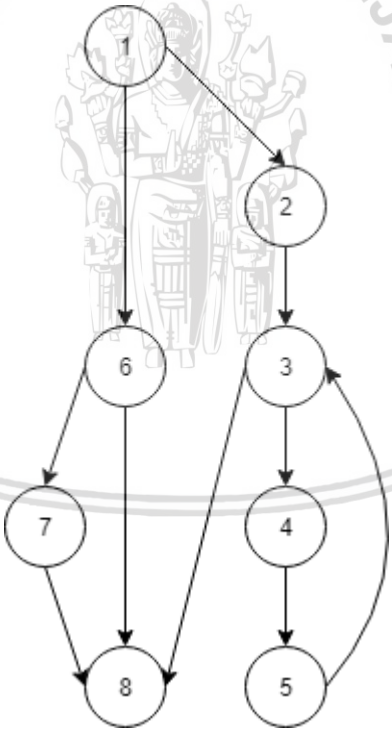
BAB 6 PENGUJIAN

6.1 Pengujian Unit

Pengujian unit skrip Hv_temp_coin_AudioLib.cs dilakukan untuk menguji algoritma pada skrip *audio* permainan. Pengujian menentukan *valid* atau tidak *valid* nya algoritma program yang diuji. Tabel 6.1 menunjukkan *Pseudocode* dari *method Update()*.

Tabel 6.1 Pengujian Unit *Update()* Skrip *Hv_temp_coin_AudioLib.cs*

No	Pseudocode
1	`If _context.IsSendHookRegistered() then`
2	`Hv_temp_coin_AudioLib.FloatMessage tempMessage;`
3	`While ((tempMessage = _context.msgQueue.GetNextMessage()) != null)`
4	`FloatReceivedCallback(tempMessage);`
5	`End while`
6	`Else If (Player_Score.count > tempcount)then`
7	`_context.SendFloatToReceiver((uint)Parameter.Seqbang, seqBang);` `tempcount++;`
8	`End if`



Gambar 6.1 *Flow Graph Update()* Skrip *Hv_temp_coin_AudioLib.cs*

Jalur Independent :

1. 1-2-3-4-5-3-8
2. 1-2-3-8
3. 1-6-7-8

4. 1-6-8

Cyclomatic Complexity (V(G)) :

1. V(G) = E+N+2 = 11-9+2 = 4
2. V(G) = P+1 = 3+1 = 4
3. V(G) = region = 4

Tabel 6.2 Hasil Pengujian Unit *Update()* Skrip *Hv_temp_coin_AudioLib.cs*

No.	No. Jalur	Data Pengujian	*Expected Result*	*Result*	Status
1	1	Jika IsSendHookRegistered() dijalankan	Mendapatkan pesan berkaitan dengan fungsionalitas *Heavy*	Mendapatkan pesan berkaitan dengan fungsionalitas *Heavy*	*Valid*
2	2	Jika IsSendHookRegistered() tidak dijalankan, Jika *count* tidak lebih besar dari *tempcount*.	*Method Update()* tidak berjalan	*Method Update()* tidak berjalan	*Valid*
3	3	Jika IsSendHookRegistered() tidak dijalankan, Jika *count* lebih besar dari *tempcount*.	*tempcount* akan menyimpan angka sebelumnya dan fungsionalitas *Heavy* untuk merubah musik akan berjalan	*tempcount* akan menyimpan angka sebelumnya dan fungsionalitas *Heavy* untuk merubah musik akan berjalan	*Valid*
4	4	Jika IsSendHookRegistered() tidak dijalankan.	*Method Update()* tidak berjalan.	*Method Update()* tidak berjalan.	*Valid*

6.2 Pengujian Validasi

Tabel 6.3 Pengujian Validasi Musik Latar Menu Utama

Nama Kasus Uji	Musik Latar Menu Utama
Prosedur	1. Pemain menjalankan permainan. 2. Pemain mengakses menu utama di dalam permainan.
Ekspetasi	Musik latar pada menu utama berjalan dan berulang.
Hasil Akhir	Musik latar menu utama dapat berjalan sesuai dengan nada yang ditetapkan serta dapat berulang.
Status	*Valid*

Tabel 6.4 Pengujian Validasi Efek Suara

Nama Kasus Uji	Efek Suara
Prosedur	1. Pemain menjalankan permainan. 2. Pemain telah mengakses menu utama di dalam permainan dan menekan tombol "*play*". 3. Pemain mengontrol karakter dan melewati rintangan. 4. Pemain mengambil koin yang tersedia.
Ekspetasi	Nada musik akan berubah ketika koin terambil.
Hasil Akhir	Nada musik berubah ketika koin yang tersedia di dalam permainan diambil oleh karakter.
Status	*Valid*

Tabel 6.5 Musik Latar Suara Angin

Nama Kasus Uji	Musik Latar Suara Angin
Prosedur	1. Pemain sedang menjalankan permainan. 2. Pemain mengontrol karakter dan melewati rintangan hingga sampaii ke titik lantai dua dalam permainan. 3. Pemain melewati aset pohon.
Ekspetasi	Efek suara akan berhenti dan tergantikan dengan musik latar suara angin.
Hasil Akhir	Ketika pemain melewati aset pohon, efek suara berhenti dan terganti dengan musik latar suara angin.
Status	*Valid*

6.3 Tabel Perbandingan

Tabel 6.6 Perbandingan Jenis Suara

Jenis Suara		Ukuran *Audio*	Jumlah Dokumen	Total Waktu
Efek Suara	*Audio Mp3*	300 kB	5	3 x 5 Detik
	Audio Skrip	14 kB	1	-
	Pure Data	3 kB	1	-
Musik Latar Menu Utama	*Audio Mp3*	60 kB	1	3 Detik
	Audio Skrip	13 kB	1	-
	Pure Data	2 kB	1	-
Musik Latar Suara Angin	*Audio Mp3*	145 kB	1	7 Detik
	Audio Skrip	11 kB	1	-
	Pure Data	1 kB	1	-

Tabel 6.6 menunjukkan bahwa dari ketiga jenis suara, ukuran *audio* dengan format *mp3* lebih besar daripada skrip yang mengandung suatu *audio* tertentu. Dalam satu contoh yaitu efek suara, skrip yang merupakan hasil dari implementasi *Pure Data* ke *Heavy* memiliki ukuran sebesar 14 kB. Sedangkan *audio mp3* yang merupakan hasil proses perekaman, memiliki ukuran sebesar 300 kB. *Pure Data* adalah perancangan yang dilakukan untuk menghasilkan jenis suara dan merupakan bentuk awal sebelum diimplementasikan untuk menghasilkan sebuah *audio* skrip.

Total waktu merupakan panjang waktu dokumen *mp3* direkam. Pada efek suara, *audio mp3* memiliki durasi rekam sebesar 3 detik pada setiap dokumennya. Terdapat 5 dokumen *audio mp3* pada efek suara, sehingga total waktu dikalikan lima.

Dalam pengembangan suara di dalam *video game*, untuk dapat membuat efek suara yang dilakukan pada penelitian ini, membutuhkan setidaknya 5 dokumen *mp3* yang berisi konten nada. Setiap nada diprogram yaitu terdapat kondisi ketika koin terambil nada akan berganti sesuai isi dokumen *mp3* satu dengan lainnya. Tabel 6.6 menunjukkan jumlah dokumen pada *audio mp3* membutuhkan 5 dokumen. Sedangkan pada *audio* skrip jumlah dokumen yang dihasilkan dapat dipangkas menjadi satu dokumen. Oleh karena itu penggunaan *audio* skrip yang merupakan hasil implementasi dari *Pure Data* lebih efisien dibandingkan dengan penggunaan *audio mp3*.

BAB 7 KESIMPULAN

7.1 Kesimpulan

Implementasi dari pemrograman visual *Pure Data* ke dalam permainan *platformer* telah berhasil dicapai, dan dapat dibuktikan dengan menghasilkan *audio* berupa efek suara, dan musik latar. *Audio* tersebut dihasilkan berdasarkan perancangan *Pure Data patch* yaitu dokumen keluaran *Pure Data*. Dengan ini dapat disimpulkan dari rumusan masalah, perancangan, implementasi, serta pengujian sebagai berikut :

1. Pemrograman visual *Pure Data* merupakan penghasil prosedural *audio* atau penghasil *audio* secara *real-time* yang memiliki jenis suara *synthesis*, dan dapat diimplementasikan ke dalam pengembangan *video game*. *Pure Data* memiliki kelebihan dibandingkan dengan jenis teknologi *audio* perekaman yang membutuhkan proses menangkap sinyal dengan menggunakan mikrofon, *mixing* suara hingga proses *mastering* untuk menjadi bentuk akhir dari *audio* tersebut.
2. Semua ukuran yang dihasilkan dari ketiga jenis suara dalam format *audio mp3*, lebih besar dibandingkan dengan skrip *audio* yang merupakan hasil dari implementasi *Pure Data* ke *Heavy*. Selain itu dengan memanfaatkan skrip *audio* dapat memangkas jumlah dokumen yang digunakan dalam pengembangan *video game*. Dapat disimpulkan bahwa penggunaan *Pure Data* lebih efisien dibandingkan dengan *audio mp3*.
3. Memanfaatkan kerangka kerja alat bantu *Heavy*, merupakan salah satu cara untuk mengimplementasikan *Pure Data* ke dalam *Unity* dengan mudah. Hanya dengan mengakses laman kerangka kerja *Heavy* pengembang dapat menghasilkan skrip *audio* dan secara langsung digunakan.
4. Dilakukan dua pengujian fungsionalitas untuk menguji *Pure Data* yang sudah diimplementasikan ke dalam permainan, yaitu pengujian unit dan validasi. Dari setiap pengujian terdapat kasus uji yang menjadi tolak ukur keberhasilan dari pengujian. Hasil yang didapatkan dari setiap pengujian adalah status *valid* pada setiap kasus ujinya.

7.2 Saran

Berikut merupakan beberapa saran untuk implementasi lanjut dari pemrograman visual *Pure Data* :

1. Dalam penelitian kali ini haya menggunakan dan memanfaatkan kerangka kerja Heavy. Terdapat cara lain untuk mengimplementasikan pemrograman visual *Pure Data*, yaitu dengan memanfaatkan *library* bernama *Libpd*. Pemrograman visual *Pure Data* dapat diimplementasikan ke dalam aplikasi ponser pintar maupun *platform* pc *Windows* menggunakan *Libpd*.

2. Terdapat *Pure Data Reference Card* yaitu referensi kartu-kartu pada *Pure Data* karangan Karim BARKATI (2010) yang dapat menjadi pedoman untuk mempelajari *Pure Data*. Dikarenakan masih banyaknya kartu-kartu *Pure Data* pada penelitian ini yang belum diimplementasikan dan dimanfaatkan. Selain musik dengan suara berjenis *synthesis* terdapat banyak efek suara maupun musik latar yang dapat dibuat dan dikembangkan sesuai dengan *Pure Data Reference Card*.

DAFTAR PUSTAKA

Ausin, D.S., Pezzarossa, L. dan Schoeberl, M., 2017. *Real-Time Audio Processing on the T-CREST Multicore Platform*. Denmark: University of Denmark.

Blesinzki, C. 2000. *The Art and Science of Level Design*. USA: Game Developers Conference 2000

Byrne, E., 2004. *Game Level Design*. USA: Charles River Media.

Farnell, A., 2007. *An introduction to procedural audio and its application in computer games*.

Gamasutra, 1996. *Tricks and Techniques for Sound Effect Design By Bobby Prince*. [online] Tersedia di: <https://www.gamecareerguide.com/features/sound_and_music/081997/sound_effect.htm> [Diakses 6 Juni 2018]

Gamasutra, 2016. *16-step Sequencer in Pure Data*. [online] Tersedia di: <https://www.gamasutra.com/blogs/AdamSporka/20161227/288411/16step_Sequencer_in_Pure_Data.php> [Diakses 6 Juni 2018]

Gormanley, S., 2013. *Audio immersion in games - a case study using an online game with background music and sounds effects*. United Kingdom: University of the West of Scotland.

Grimshaw, M., 2011. *Game Sound Technology and Player Interaction* : *Concepts and Developments*. United Kingdom : University of Bolton.

Hodgson, J., 2010. *Understanding Records*. United Kingdom : Bloomsburry Publishing.

Lindu, M., 2017. Pengembangan *Game Platformer* 2D Menggunakan Teknik *Projection Mapping*. Jurnal Pengembangan Teknologi Informasi dan Ilmu Komputer. Malang : Universitas Brawijaya.

Minkkinen, T., 2016. *Basics of Platform Games*. Belanda: University of Applied Sciences.

Novation, 2018. *Launch Control*. [image online] Tersedia di: <https://us.novationmusic.com/launch/launch-control#> [Diakses 6 Juni 2018]

Peerdeman, P., 2010. *Sound and Music in Games*. Belanda: VU Amsterdam.

Pure Data, 2006. *Pd Documentation*. [image online] Tersedia di: <https://puredata.info/docs/manuals/pd/x2.htm#s1> [Diakses 6 Juni 2018]

Russ, M., 2009. *Sound Synthesis and Sampling Third Edition*. Oxford: Elsevier Ltd..

Smith, G., Cha, M. dan Whitehead, J., 2008. *A Framework fr Analysis of 2D Platformer Levels*. California: University of California.

Widodo, T.W., 2006. Komputer dan Pengetahuan Program Aplikasi Musik Komputer. Yogyakarta: Institut Seni Indonesia Yogyakarta

Wilcox, D., 2016. *PdParty: An iOS Computer Music Platform using libpd*. New York: Pure Data Conference.

Wei, T. dan Zheng, H., 2011. *Sound Effect of Physical Engine in Game Design*. Nanjing, Jiangsu, China: Xiaozhuang University.

repository.ub.ac.id
UNIVERSITAS BRAWIJAYA